Randomized Computations on Large Data Sets: Tight Lower Bounds

– Full Version –

Martin Grohe André Hernich Nicole Schweikardt
Institut für Informatik, Humboldt-Universität, Berlin, Germany
{grohe | hernich | schweika}@informatik.hu-berlin.de

ABSTRACT

We study the randomized version of a computation model (introduced in [9, 10]) that restricts random access to external memory and internal memory space. Essentially, this model can be viewed as a powerful version of a data stream model that puts no cost on sequential scans of external memory (as other models for data streams) and, in addition, (like other external memory models, but unlike streaming models), admits several large external memory devices that can be read and written to in parallel.

We obtain tight lower bounds for the decision problems set equality, multiset equality, and checksort. More precisely, we show that any randomized one-sided-error bounded Monte Carlo algorithm for these problems must perform $\Omega(\log N)$ random accesses to external memory devices, provided that the internal memory size is at most $O(\sqrt[4]{N}/\log N)$, where N denotes the size of the input data.

From the lower bound on the set equality problem we can infer lower bounds on the worst case data complexity of query evaluation for the languages XQuery, XPath, and relational algebra on streaming data. More precisely, we show that there exist queries in XQuery, XPath, and relational algebra, such that any (randomized) Las Vegas algorithm that evaluates these queries must perform $\Omega(\log N)$ random accesses to external memory devices, provided that the internal memory size is at most $O(\sqrt[4]{N}/\log N)$.

Categories and Subject Descriptors

F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes; F.1.1 [**Computation by Abstract Devices**]: Models of Computation

General Terms

Theory, Languages

Keywords

complexity, data streams / real-time data, query processing / query optimization, semi-structured data, XML

1. INTRODUCTION

Today's hardware technology provides a hierarchy of storage media from tapes and disks at the bottom through main memory and (even on-CPU) memory caches at the top. Storage media from different levels of this memory hierarchy considerably differ in price, storage size, and access time. Currently, the most pronounced performance and price (and consequently also size) gap is between main memory and the next-lower level in the memory hierarchy, usually magnetic disks which have to rely on comparably slow, mechanical, physically moving parts. One often refers to the upper layers above this gap by *internal memory* and the lower layers of the memory hierarchy by *external memory*. The technological reality is such that the time for accessing a given bit of information in external memory is five to six orders of magnitude larger than the time required to access a bit in internal memory. Apart from this, concerning external memory, *random accesses* (which involve moving the disk head to a particular location) are significantly more expensive than *sequential scans*.

Modern software and database technology uses clever heuristics to minimize the number of accesses to external memory and to prefer *streaming* over *random accesses* to external memory. There has also been a wealth of research on the design of so-called *external memory algorithms* (cf., e.g. [16, 18, 13]). The classes considered in *computational complexity theory*, however, usually do not take into account the existence of different storage media. In [9, 10], we introduced a formal model for such a scenario. The two most significant cost measures in our setting are the number of random accesses to external memory and the size of the internal memory. Our model is based on a standard multi-tape Turing machine. Some of the tapes of the machine, among them the input tape, represent the external memory. They are unrestricted in size, but access to these tapes is restricted by allowing only a certain number $r(N)$ (where N denotes the input size) of reversals of the head directions. This may be seen as a way of (a) restricting the number of sequential scans and (b) restricting random access to these tapes, because each random access can be simulated by moving the head to the desired position on a tape, which involves at most two head reversals. The remaining tapes of the Turing machine represent the internal memory. Access to these internal memory tapes (i.e., the number of head reversals) is unlimited, but their size is bounded by a parameter $s(N)$. We let $\mathrm{ST}(r(N), s(N), O(1))$ denote the class of all problems that can be solved on such an $\big(r(N), s(N), O(1)\big)$-bounded Turing machine, i.e., a Turing machine with an arbitrary number of external memory tapes which, on inputs of size N, performs less than $r(N)$ head reversals on the external memory tapes, and uses at most space $s(N)$ on the internal memory tapes.

The astute reader who wonders if it is realistic to assume that the external memory tapes can be read in *both* directions (which disks

cannot so easily) and that a sequential scan of an entire external memory tape accounts for only one head reversal (and thus seems unrealistically cheap) be reminded that this paper's main goal is not to design efficient external memory algorithms but, instead, to prove *lower* bounds. Thus, considering a rather powerful computation model makes our lower bound results only stronger.

In the present paper, we prove lower bounds for *randomized* computations (i.e., computations where in each step a coin may be tossed to determine the next configuration) in a scenario with several storage media. To this end, we introduce the complexity class $\mathrm{RST}(r(N), s(N), O(1))$, which consists of all decision problems that can be solved by an $\big(r(N), s(N), O(1)\big)$-bounded randomized Turing machine with one-sided bounded error, where no false positive answers are allowed and the probability of false negative answers is at most 0.5 (in the literature, such randomized algorithms are often called *one-sided-error Monte Carlo algorithms*, cf. [12]). To also deal with computation problems where an output (other than just a yes/no answer) has to be generated, we write *LasVegas*-$\mathrm{RST}\big(r(N), s(N), O(1)\big)$ to denote the class of all functions f for which there exists an $\big(r(N), s(N), O(1)\big)$-bounded randomized Turing machine that, for every input word w, (a) always produces either the correct output $f(w)$ on one of its external memory tapes or gives the answer *"I don't know"* and (b) gives the answer *"I don't know"* with probability at most 0.5 (in the literature, such randomized algorithms are sometimes called *Las Vegas algorithms*, cf. [12]).

Contributions: Our first main result is a lower bound for three natural decision problems: The *set equality problem* and the *multiset equality problem* ask whether two given (multi)sets of strings are equal, and the *checksort problem* asks, given two sequences of strings, whether the second is a sorted version of the first.
We show (Theorem 6) that neither problem is contained in $\mathrm{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$. This lower bound turns out to be *tight* in the following senses:

- If the number of sequential scans (i.e., head reversals) increases from $o(\log N)$ to $O(\log N)$, then each of the three problems can be solved with only constant internal memory and without using randomization. In other words (see Corollary 7), the (multi)set equality problem and the checksort problem belong to $\mathrm{ST}(O(\log N), O(1), O(1))$.
- When using randomization with the complementary one-sided error model, i.e., machines where no false negative answers are allowed and the probability of false positive answers is at most 0.5, then the *multiset equality* problem can be solved with just two sequential scans of the input (and without ever writing to external memory), and internal memory of size $O(\log N)$. In other words (Theorem 8(a)), the multiset equality problem belongs to co-$\mathrm{RST}(2, O(\log N), 1)$.
- When using nondeterministic machines, then (multi)set equality and checksort can be solved with three sequential scans on two external memory tapes and internal memory of size $O(\log N)$. In other words (Theorem 8(b)), the (multi)set equality problem and the checksort problem belong to $\mathrm{NST}(3, O(\log N), 2)$.

As a consequence, we obtain a separation between the deterministic, the randomized, and the nondeterministic $\mathrm{ST}(\cdots)$ classes (Corollary 9).

Our lower bound for the checksort problem, in particular, implies that the *sorting problem* (i.e., the problem of sorting a sequence of input strings) does not belong to the complexity class *LasVegas*-$\mathrm{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$ and thus generalizes the main result of [10] to randomized computations.

Our lower bound for the set equality problem leads to the following lower bounds on the worst case data complexity of database query evaluation problems in a streaming context:

- There is an *XQuery* query Q such that the problem of evaluating Q on an input XML document stream of length N does *not* belong to the class *LasVegas*-$\mathrm{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$ (Theorem 12).

Speaking informally, this means that, no matter how many external memory devices (of arbitrarily large size) are available, as long as the internal memory is of size at most $O(\frac{\sqrt[4]{N}}{\log N})$, every randomized algorithm that produces the correct query result with probability at least 0.5 will perform $\Omega(\log N)$ random accesses to external memory. We obtain analogous results for *relational algebra* queries and for the node-selecting XML query language *XPath*:

- There is a *relational algebra* query Q such that the problem of evaluating Q on a stream consisting of the tuples of the input database relations does *not* belong to the complexity class *LasVegas*-$\mathrm{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$, where N denotes the total size of the input database relations. Furthermore, this bound is tight with respect to the number of random accesses to external memory, as the data complexity of every relational algebra query belongs to $\mathrm{ST}(O(\log N), O(1), O(1))$ (Theorem 11).
- There is an *XPath* query Q such that the problem of filtering an input XML document stream with Q (i.e., checking whether at least one node of the document matches the query) does *not* belong to the class co-$\mathrm{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$ (Theorem 13).

This means that there is an *XPath* query Q such that, no matter how many external memory devices (of arbitrarily large size) are available, as long as the internal memory is of size at most $O(\frac{\sqrt[4]{N}}{\log N})$, every randomized algorithm which accepts every input document that matches Q, and which rejects documents not matching Q with probability ≥ 0.5, will perform $\Omega(\log N)$ random accesses to external memory.

Related Work: Obviously, our model is related to the *bounded reversal Turing machines*, which have been studied in classical complexity theory (see, e.g., [19, 7]). However, in bounded reversal Turing machines, the number of head reversals is limited on *all* tapes, whereas in our model there is no such restriction on the internal memory tapes. This makes our model considerably stronger, considering that in our lower bound results we allow internal memory size that is polynomially related to the input size. Furthermore, to our best knowledge, all lower bound proofs previously known for reversal complexity classes on multi-tape Turing machines go back to the space hierarchy theorem (cf., e.g., [17, 7]) and thus rely on diagonalization arguments, and apply only to classes with $\omega(\log N)$ head reversals. In particular, these lower bounds do not include the checksort problem and the (multi)set equality problem, as these problems can be solved with $O(\log N)$ head reversals.

In the classical *parallel disk model* for external memory algorithms (see, e.g., [18, 13, 16]), the cost measure is simply the number of bits read from external memory divided by the page size. Several refinements of this model have been proposed to include a distinction between *random access* and *sequential scans* of the external memory, among them Arge and Bro Miltersen's *external*

memory Turing machines [3]. We note that their notion of external memory Turing machines significantly differs from ours, as their machines only have a single external memory tape and process inputs that consist of a *constant* number m of input strings. Strong lower bound results (in particular, for different versions of the *sorting problem*) are known for the parallel disk model (see [18] for an overview) as well as for Arge and Bro Miltersen's external memory Turing machines [3]. However, to the best of our knowledge, all these lower bound proofs heavily rely on the assumption that the input data items (e.g., the strings that are to be sorted) are *indivisible* and that at any point in time, the external memory consists, in some sense, of a permutation of the input items. We emphasize that the present paper's lower bound proofs do not rely on such an indivisibility assumption.

Strong lower bounds for a number of problems are known in the context of *data streams* and for models which permit a small number of sequential scans of the input data, but no auxiliary external memory (that is, the version of our model with no extra external memory tapes apart from the input tape) [15, 2, 11, 4, 16, 5, 6, 1, 9]. All these lower bounds are obtained by communication complexity. Note that in the presence of at least two external memory tapes, communication between remote parts of memory is possible by simply copying data from one tape to another and then rereading both tapes in parallel. These communication abilities of our model spoil any attempt to prove lower bounds via communication complexity, which is the tool of choice both for computation models permitting few scans but no auxiliary external memory, and for 1-tape Turing machines.

The deterministic $\mathrm{ST}(\cdots)$-classes were introduced in [10, 9]. In [9] we studied those classes where only a *single* external memory tape is available and used methods from communication complexity to obtain lower bounds for these classes. The main result of [10] was a lower bound for the *sorting problem* concerning the deterministic $\mathrm{ST}(\cdots)$-classes with an *arbitrary* number of external memory tapes. An important tool for proving this bound was to introduce deterministic *list machines* as an intermediate machine model. An overview of the methods used and the results obtained in [9, 10] was given in [8]. The present paper builds on [10], as it considers $\mathrm{ST}(\cdots)$-classes with an *arbitrary* number of external memory tapes and it uses *list machines* as a key tool for proving lower bound results. However, the results presented here go significantly beyond those obtained in [10]. Here we obtain lower bounds for *decision* problems in the *randomized* versions of the model. The main result of [10] is that the sorting problem does not belong to $\mathrm{ST}(o(\log N), O(\frac{\sqrt[5]{N}}{\log N}), O(1))$, and the proof given there heavily relies on the fact that the machines are *deterministic* and the *output* of the sorting problem cannot be generated within the given resource bounds. In contrast to the present paper's approach, the proof method of [10] neither works for decision problems, i.e. problems where no output is generated, nor for randomized computations. Finally, let us remark that the main result of [10] can be obtained as an immediate corollary of the present paper's lower bound for the checksort problem.

Organization: After introducing the deterministic, the nondeterministic, and the randomized $\mathrm{ST}(\cdots)$ classes in Section 2, we formally state our main lower bounds for decision problems in Section 3. In Section 4 we use these results to derive lower bounds on the data complexity of query evaluation for the languages *XQuery*, *XPath*, and *relational algebra*. The subsequent sections are devoted to the proof of the lower bound on the decision problems *(multi)set equality* and *checksort*: In Section 5, 6, and 7 we introduce randomized *list machines*, show that randomized Turing machines can be simulated by randomized list machines, and prove that randomized list machines can neither solve the *(multi)set equality problem* nor the *checksort problem*. Afterwards, in Section 8 we transfer these results from list machines to Turing machines. We close with a few concluding remarks and open problems in Section 9.

The present paper is the full version of the extended abstract published in the proceedings of the 25th ACM Sigact-Sigart Symposium on Principles of Database Systems (PODS'06).

2. COMPLEXITY CLASSES

We write $\mathbb{N}$ to denote the set of natural numbers (that is, nonnegative integers).

As our basic model of computation, we use standard multi-tape nondeterministic Turing machines (NTMs, for short); cf., e.g., [17]. The Turing machines we consider will have $t+u$ tapes. We call the first t tapes *external memory tapes* (and think of them as representing t disks). We call the other u tapes *internal memory tapes*. The first tape is always viewed as the input tape.

Without loss of generality we assume that our Turing machines are normalized in such a way that in each step at most one of its heads moves to the left or to the right.

Let T be an NTM and ρ a finite run of T. Let $i \geq 1$ be the number of a tape. We use $\mathrm{rev}(\rho, i)$ to denote the number of times the head on tape i changes its direction in the run ρ. Furthermore, we let $\mathrm{space}(\rho, i)$ be the number of cells of tape i that are used by ρ.

Definition 1 ((r,s,t)-bounded TM). Let $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$. A (nondeterministic) Turing machine T is *(r,s,t)-bounded*, if every run ρ of T on an input of length N (for arbitrary $N \in \mathbb{N}$) satisfies the following conditions: (1) ρ is finite, (2) $1 + \sum_{i=1}^{t} \mathrm{rev}(\rho, i) \leq r(N)$, and[1] (3) $\sum_{i=t+1}^{t+u} \mathrm{space}(\rho, i) \leq s(N)$, where $t+u$ is the total number of tapes of T. ⊣

Definition 2 ($\mathrm{ST}(\cdots)$ and $\mathrm{NST}(\cdots)$ classes).
Let $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$. A decision problem belongs to the class $\mathrm{ST}(r,s,t)$ (resp., $\mathrm{NST}(r,s,t)$), if it can be decided by a deterministic (resp., nondeterministic) (r,s,t)-bounded Turing machine. ⊣

Note that we put no restriction on the running time or the space used on the first t tapes of an (r,s,t)-bounded Turing machine. The following lemma shows that these parameters cannot get too large.

Lemma 3 ([10]). *Let $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$, and let T be an (r,s,t)-bounded NTM. Then for every run $\rho = (\rho_1, \ldots, \rho_\ell)$ of T on an input of size N we have $\ell \leq N \cdot 2^{O(r(N) \cdot (t + s(N)))}$ and thus $\sum_{i=1}^{t} \mathrm{space}(\rho, i) \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$.* ⊣

In [10], the lemma has only been stated and proved for deterministic Turing machines, but it is obvious that the same proof also applies to nondeterministic machines (to see this, note that, by definition, *every* run of an (r,s,t)-bounded Turing machine is finite).

In analogy to the definition of *randomized* complexity classes such as the class RP of randomized polynomial time (cf., e.g., [17]), we consider the randomized versions $\mathrm{RST}(\cdots)$ and *LasVegas*-$\mathrm{RST}(\cdots)$ of the $\mathrm{ST}(\cdots)$ and $\mathrm{NST}(\cdots)$ classes. The following definition of randomized Turing machines formalizes the intuition that in each step, a coin can be tossed to determine which particular successor configuration is chosen in this step. For a configuration γ of an NTM T, we write $\mathrm{Next}_T(\gamma)$ to denote the set of all configurations γ' that can be reached from γ in a single step. Each

[1] It is convenient for technical reasons to add 1 to the number $\sum_{i=1}^{t} \mathrm{rev}(\rho, i)$ of changes of the head direction here. As defined here, $r(N)$ thus bounds the number of sequential scans of the external memory tapes rather than the number of changes of head directions.

such configuration $\gamma' \in \text{Next}_T(\gamma)$ is chosen with uniform probability, i.e., $\Pr(\gamma \to_T \gamma') = 1/|\text{Next}_T(\gamma)|$. For a run $\rho = (\rho_1, .., \rho_\ell)$, the probability $\Pr(\rho)$ that T performs run ρ is the product of the probabilities $\Pr(\rho_i \to_T \rho_{i+1})$, for all $i < \ell$. For an input word w, the probability that T accepts w (resp., that T outputs w') is defined as the sum of $\Pr(\rho)$ for all accepting runs ρ of T on input w (resp., of all runs of T on w that output w'). We say that a decision problem L is solved by a $(\frac{1}{2}, 0)$-RTM if, and only if, there is an NTM T such that every run of T has finite length, and the following is true for all input instances w: If $w \in L$, then $\Pr(T \text{ accepts } w) \geq 1/2$; if $w \notin L$, then $\Pr(T \text{ accepts } w) = 0$. Similarly, we say that a function $f : \Sigma^* \to \Sigma^*$ is computed by a *LasVegas*-RTM if, and only if, there is an NTM T such that every run of T on every input instance w has finite length and outputs either $f(w)$ or *"I don't know"*, and $\Pr(T \text{ outputs } f(w)) \geq 1/2$.

Definition 4 (RST$(\cdots)$ and *LasVegas*-RST$(\cdots)$).
Let $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$.

(a) A decision problem L belongs to the class $\text{RST}(r,s,t)$, if it can be solved by a $(\frac{1}{2}, 0)$-RTM that is (r,s,t)-bounded.

(b) A function $f : \Sigma^* \to \Sigma^*$ belongs to *LasVegas*-$\text{RST}(r,s,t)$, if it can be solved by a *LasVegas*-RTM that is (r,s,t)-bounded. ⊣

As a straightforward observation one obtains:

Proposition 5. *For all $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$,*
$\text{ST}(r,s,t) \subseteq \text{RST}(r,s,t) \subseteq \text{NST}(r,s,t)$. ⊣

For classes R and S of functions we let $\text{ST}(R,S,t) := \bigcup_{r \in R, s \in S} \text{ST}(r,s,t)$ and $\text{ST}(R,S,O(1)) := \bigcup_{t \in \mathbb{N}} \text{ST}(R,S,t)$. Analogous notations are used for the $\text{NST}(\cdots)$, $\text{RST}(\cdots)$, and *LasVegas*-$\text{RST}(\cdots)$ classes, too.

As usual, for every (complexity) class C of decision problems, co-C denotes the class of all decision problems whose *complements* belong to C. Note that the $\text{RST}(\cdots)$-classes consist of decision problems that can be solved by randomized algorithms that allow a moderate number of false negatives, but no false positives. In contrast to this, the co-$\text{RST}(\cdots)$-classes consist of problems that can be solved by randomized algorithms that allow a moderate number of false positives, but no false negatives.

From Lemma 3, one immediately obtains for all functions r, s with $r(N) \cdot s(N) \in O(\log N)$ that $\text{ST}(r,s,O(1)) \subseteq$ PTIME, $\text{RST}(r,s,O(1)) \subseteq$ RP, and $\text{NST}(r,s,O(1)) \subseteq$ NP (where PTIME, RP, and NP denote the class of problems solvable in polynomial time on deterministic, randomized, and nondeterministic Turing machines, respectively).

3. LOWER BOUNDS FOR DECISION PROBLEMS

Our first main result is a lower bound for the *(multi)set equality problem* as well as for the *checksort problem*. The *(multi)set equality problem* asks if two given (multi)sets of strings are the same. The *checksort problem* asks for two input lists of strings whether the second list is the lexicographically sorted version of the first list. We encode inputs as strings over the alphabet $\{0, 1, \#\}$. Formally, the *(multi)set equality* and the *checksort* problem are defined as follows: The input instances of each of the three problems are

Instance: $v_1\#\cdots\#v_m\#v'_1\#\cdots\#v'_m\#$,
where $m \geq 0$, and $v_i, v'_i \in \{0,1\}^*$ (for all $i \leq m$)

and the task is to decide the following:

SET-EQUALITY *problem:*
Decide if $\{v_1, \ldots, v_m\} = \{v'_1, \ldots, v'_m\}$.

MULTISET-EQUALITY *problem:*
Decide if the multisets $\{v_1, \ldots, v_m\}$ and $\{v'_1, \ldots, v'_m\}$ are equal (i.e., they contain the same elements with the same multiplicities).

CHECK-SORT *problem:*
Decide if $v'_1, \ldots, v'_m$ is the lexicographically sorted (in ascending order) version of $v_1, \ldots, v_m$.

For an instance $v_1\#\cdots v_m\#v'_1\#\cdots v'_m\#$ of the above problems, we usually let $N = 2m + \sum_{i=1}^{m}(|v_i| + |v'_i|)$ denote the size of the input. Furthermore, in our proofs we will only consider instances where all the v_i and v'_i have the same length n, so that $N = 2m \cdot (n+1)$.

The present paper's technically most involved result is the following lower bound:

Theorem 6. *Let $r, s : \mathbb{N} \to \mathbb{N}$ such that $r(N) \in o(\log N)$ and $s(N) \in o\left(\sqrt[4]{N}/r(N)\right)$. Then, none of the problems* CHECK-SORT, SET-EQUALITY, MULTISET-EQUALITY *belongs to the class* $\text{RST}(r(N), s(N), O(1))$. ⊣

Sections 5–8 are devoted to the proof of Theorem 6. The proof uses an intermediate computation model called *list machines* and proceeds by (1) showing that randomized Turing machine computations can be simulated by randomized list machines that have the same acceptance probabilities as the given Turing machines and (2) proving a lower bound for (MULTI)SET-EQUALITY and CHECK-SORT on randomized list machines.

By applying the reduction used in [10, Theorem 9], we obtain that the lower bound of Theorem 6 also applies for the "SHORT" versions of (MULTI)SET-EQUALITY and CHECK-SORT, i.e., the restrictions of these problems to inputs of the form $v_1\#\cdots v_m\# v'_1\#\cdots v'_m\#$, where each v_i and v'_i is a 0-1-string of length at most $c \cdot \log m$, and c is an arbitrary constant ≥ 2. By using the standard *merge sort* algorithm, one easily obtains that the "SHORT" versions of (MULTI)SET-EQUALITY and CHECK-SORT belong to $\text{ST}(O(\log N), O(\log N), 3)$. Moreover, in [7, Lemma 7] it has been shown that the (general) sorting problem can be solved by an $\big(O(\log N), O(1), 2\big)$-bounded deterministic Turing machine. As an immediate consequence, we obtain:

Corollary 7. SET-EQUALITY, MULTISET-EQUALITY, CHECK-SORT, *and their* "SHORT" *versions, are in* $\text{ST}(O(\log N), O(1), 2)$, *but not in* $\text{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. ⊣

A detailed proof can be found in Appendix E.
As a further result, we show that

Theorem 8. *(a)* MULTISET-EQUALITY *belongs to* $\text{co-RST}(2, O(\log N), 1) \subseteq \text{co-NST}(2, O(\log N), 1)$.

(b) Each of the problems MULTISET-EQUALITY, CHECK-SORT, SET-EQUALITY *belongs to* $\text{NST}(3, O(\log N), 2)$. ⊣

Proof: (a) We apply fairly standard *fingerprinting techniques* and show how to implement them on a $(2, O(\log N), 1)$-bounded randomized Turing machine. Consider an instance $v_1\#\ldots\#v_m\#v'_1\#\ldots\# v'_m\#$ of the MULTISET-EQUALITY problem. For simplicity, let us assume that all the v_i and v'_j have the same length n. Thus the input size N is $2 \cdot m \cdot (n+1)$. We view the v_i and v'_i as integers in $\{0, \ldots, 2^n - 1\}$ represented in binary.

We use the following algorithm to decide whether the multisets $\{v_1, \ldots, v_m\}$ and $\{v'_1, \ldots, v'_m\}$ are equal:

(1) During a first sequential scan of the input, determine the input parameters n, m, and N.

(2) Choose a prime $p_1 \leq k := m^3 \cdot n \cdot \log(m^3 \cdot n)$ uniformly at random.

(3) Choose an arbitrary prime p_2 such that $3k < p_2 \leq 6k$. Such a prime exists by Bertrand's postulate.

(4) Choose $x \in \{1, \ldots, p_2 - 1\}$ uniformly at random.

(5) For $1 \leq i \leq m$, let $e_i = (v_i \bmod p_1)$ and $e'_i = (v'_i \bmod p_1)$. If

$$\sum_{i=1}^{m} x^{e_i} \;\equiv\; \sum_{i=1}^{m} x^{e'_i} \bmod p_2 \qquad (1)$$

then accept, else reject.

Let us first argue that the algorithm is correct (for sufficiently large m, n): Clearly, if the multisets $\{v_1, \ldots, v_m\}$ and $\{v'_1, \ldots, v'_m\}$ are equal then the algorithm accepts. On the other hand, if they are distinct, the probability that the multisets $\{e_1, \ldots, e_m\}$ and $\{e'_1, \ldots, e'_m\}$ are equal is $O(1/m)$. This is due to the following.

CLAIM 1. *Let $n, m \in \mathbb{N}$, $k = m^3 \cdot n \cdot \log(m^3 \cdot n)$, and $0 \leq v_1, .., v_m, v'_1, .., v'_m < 2^n$. Then for a prime $p \leq k$ chosen uniformly at random, $\Pr(\exists i, j \leq m$ with $v_i \neq v'_j$ and $v_i \equiv v'_j \bmod p) \leq O(1/m)$.*

Proof: We use the following well-known result (see, for example, Theorem 7.5 of [14]): Let $n, \ell \in \mathbb{N}$, $k = \ell \cdot n \cdot \log(\ell \cdot n)$, and $0 < x < 2^n$. Then for a prime $p \leq k$ chosen uniformly at random,

$$\Pr(x \equiv 0 \bmod p) \;\leq\; O\left(\tfrac{1}{\ell}\right).$$

The claim then follows if we apply this result with $\ell = m^3$ simultaneously to the at most m^2 numbers $x = v_i - v'_j$ with $v_i \neq v'_j$. □

To proceed with the proof of Theorem 8(a), suppose that the two multisets are distinct. Then the polynomial

$$q(X) \;=\; \sum_{i=1}^{m} X^{e_i} - \sum_{i=1}^{m} X^{e'_i}$$

is nonzero. Note that all coefficients and the degree of $q(X)$ are at most $k < p_2$. We view $q(X)$ as a polynomial over the field $\mathbb{F}_{p_2}$. As a nonzero polynomial of degree at most p_1, it has at most p_1 zeroes. Thus the probability that $q(x) = 0$ for the randomly chosen $x \in \{1, \ldots, p_2 - 1\}$ is at most $p_1/(p_2 - 1) \leq 1/3$. Therefore, if the multisets $\{e_1, \ldots, e_m\}$ and $\{e'_1, \ldots, e'_m\}$ are distinct, the algorithm accepts with probability at most $1/3$, and the overall acceptance probability is at most

$$\tfrac{1}{3} + O\left(\tfrac{1}{m}\right) \leq \tfrac{1}{2}$$

for sufficiently large m. This proves the correctness of the algorithm.

Let us now explain how to implement the algorithm on a $(2, O(\log N), 1)$-bounded randomized Turing machine. Note that the binary representations of the primes p_1 and p_2 have length $O(\log N)$. The standard arithmetical operations can be carried out in linear space on a Turing machine. Thus with numbers of length $O(\log N)$, we can carry out the necessary arithmetic on the internal memory tapes of our $(2, O(\log N), 1)$-bounded Turing machine.

To choose a random prime p_1 in step (2), we simply choose a random number $\leq k$ and then test if it is prime, which is easy in linear space. If the number is not prime, we repeat the procedure, and if we do this sufficiently often, we can find a random prime with high probability. Steps (3) and (4) can easily be carried out in internal memory. To compute the number e_i in step (5), we proceed as follows: Suppose the binary representation of v_i is $v_{i,(n-1)} \ldots v_{i,0}$, where $v_{i,0}$ is the least significant bit. Observe that

$$e_i \;=\; \big(\big(\sum_{j=0}^{n-1} 2^j \cdot v_{i,j}\big) \bmod p_1\big).$$

We can evaluate this sum sequentially by taking all terms modulo p_1; this way we only have to store numbers smaller than p_1. This requires one sequential scan of v_i and no head reversals.

To evaluate the polynomial $\sum_{i=1}^{m} x^{e_i}$ modulo p_2, we proceed as follows: Let $t_i = (x^{e_i} \bmod p_1)$ and $s_i = ((\sum_{j=1}^{i} t_i) \bmod p_1)$. Again we can compute the sum sequentially by computing e_i, t_i, and $s_i = ((s_{i-1} + t_i) \bmod p_1)$ for $i = 1, \ldots, m$. We can evaluate $\sum_{i=1}^{m} x^{e'_i}$ analogously and then test if (1) holds. This completes the proof of part (a) of Theorem 8.

(b): Let w be an input of length N, $w := v_1 \# v_2 \# \ldots \# v_m \# v'_1 \# v'_2 \# \ldots \# v'_m \#$. Note that the multisets $\{v_1, \ldots, v_m\}$ and $\{v'_1, \ldots, v'_m\}$ are equal if and only if there is a permutation π of $\{1, \ldots, m\}$ such that for all $i \in \{1, \ldots, m\}$, $v_i = v'_{\pi(i)}$. The idea is to "guess" such a permutation π (suitably encoded as a string over $\{0, 1, \#\}$), to write sufficiently many copies of the string $u := \pi \# w$ onto the first tape, and finally solve the problem by comparing v_i and $v'_{\pi(i)}$ bitwise, where in each step we use the next copy of u.

A $(3, O(\log N), 2)$-bounded nondeterministic Turing machine M can do this as follows. In a forward scan, it nondeterministically writes a sequence $u_1, u_2, \ldots, u_\ell$ of $\ell := m + N \cdot m$ many strings on its first and on its second tape, where

$$u_i := \pi_{i,1} \# \ldots \# \pi_{i,m} \# v_{i,1} \# \ldots \# v_{i,m} \# v'_{i,1} \# \ldots \# v'_{i,m} \#$$

for binary numbers $\pi_{i,j}$ from $\{1, \ldots, m\}$, and bit strings $v_{i,j}$ and $v'_{i,j}$ of length at most N. While writing the first $N \cdot m$ strings, it ensures that for every $i \in \{1, \ldots, N \cdot m\}$, either $v_{i,\lceil i/N \rceil}$ and $v'_{i,\pi_{i,\lceil i/N \rceil}}$ coincide on bit $((i-1) \bmod N) + 1$, or that both strings have no such bit at all. While writing the last m strings, it ensures that for all $i \in \{1, \ldots, m\}$ and $j \in \{i+1, \ldots, m\}$, $\pi_{N \cdot m + i, i} \neq \pi_{N \cdot m + i, j}$. Finally, M checks in a backward scan of both external memory tapes that $u_i = u_{i-1}$ for all $i \in \{2, \ldots, \ell\}$, and that $v_{1,j} = v_j$ and $v'_{1,j} = v'_j$ for all $j \in \{1, \ldots, m\}$.

The SET-EQUALITY problem can be solved in a similar way.

Deciding CHECK-SORT is very similar: the machine additionally has to check that v'_i is smaller than or equal to v'_j for all $j \in \{i+1, \ldots, m\}$. This can be done, e.g., by writing $N \cdot \sum_{i=1}^{m-1} i$ additional copies of u, and by comparing v'_i and v'_j bitwise on these strings for each i and $j \in \{i+1, \ldots, m\}$. □

Theorems 6 and 8, in particular, immediately lead to the following separations between the deterministic, randomized, and nondeterministic ST$(\cdots)$ classes:

Corollary 9. *Let $r, s : \mathbb{N} \to \mathbb{N}$ with $r(N) \in o(\log N)$ and $s(N) \in o\left(\sqrt[4]{N}/r(N)\right) \cap \Omega(\log N)$. Then,*

(a) $\mathrm{RST}(O(r), O(s), O(1)) \neq \text{co-RST}(O(r), O(s), O(1))$,

(b) $\mathrm{ST}(O(r), O(s), O(1)) \subsetneq \mathrm{RST}(O(r), O(s), O(1)) \subsetneq \mathrm{NST}(O(r), O(s), O(1))$.

The (straightforward) proof can be found in Appendix E.

Let us note that the lower bound of Theorem 6 for the problem CHECK-SORT in particular implies the following generalization of the main result of [10] to randomized computations:

Corollary 10. *The sorting problem (i.e., the problem of sorting a sequence of input strings) does not belong to the class LasVegas-*$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. ⊣

The (straightforward) proof can be found in Appendix E.

4. LOWER BOUNDS FOR QUERY EVALUATION

Our lower bound for the SET-EQUALITY problem (Theorem 6) leads to the following lower bounds on the worst case data com-

plexity of database query evaluation problems in a streaming context:

Theorem 11 (Tight Bound for Relational Algebra).

(a) For every relational algebra query Q, the problem of evaluating Q on a stream consisting of the tuples of the input database relations can be solved in $\mathrm{ST}(O(\log N), O(1), O(1))$.

(b) There exists a relational algebra query Q' such that the problem of evaluating Q' on a stream of the tuples of the input database relations does not belong to the class LasVegas-$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. ⊣

Proof: (a): It is straightforward to see that for every relational algebra query Q there exists a number c_Q such that Q can be evaluated within c_Q sequential scans and sorting steps. Every sequential scan accounts for a constant number of head reversals and constant internal memory space. Each sorting step can be accomplished using the sorting method of [7, Lemma 7] (which is a variant of the merge sort algorithm) with $O(\log N)$ head reversals and constant internal memory space. Since the number c_Q of necessary sorting steps and scans is constant (i.e., only depends on the query, but not on the input size N), the query Q can be evaluated by an $(O(\log N), O(1), O(1))$-bounded deterministic Turing machine.

(b): Consider the relational algebra query

$$Q' \;:=\; (R_1 - R_2) \cup (R_2 - R_1)$$

which computes the symmetric difference of two relations R_1 and R_2. Note that the query result is empty if, and only if, $R_1 = R_2$. Therefore, any algorithm that evaluates Q' solves, in particular, the SET-EQUALITY problem. Hence, if Q' could be evaluated in *LasVegas*-$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, then SET-EQUALITY could be solved in $\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, contradicting Theorem 6. □

We also obtain lower bounds on the worst case data complexity of evaluating *XQuery* and *XPath* queries against XML document streams:

Theorem 12 (Lower Bound for XQuery). *There is an* XQuery *query Q such that the problem of evaluating Q on an input XML document stream of length N does not belong to the class LasVegas-*$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. ⊣

Theorem 13 (Lower Bound for XPath). *There is an* XPath *query Q such that the problem of filtering an input XML document stream with Q (i.e., checking whether at least one node of the document matches the query) does not belong to the class* co-$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. ⊣

For proving the Theorems 12 and 13, we represent an instance $x_1\#\cdots\#x_m\#y_1\cdots\#y_m\#$ of the SET-EQUALITY problem by an XML document of the form

```
<instance>
  <set1>
    <item> <string> x1 </string> </item>
      ...
    <item> <string> xm </string> </item>
  </set1>
  <set2>
    <item> <string> y1 </string> </item>
      ...
    <item> <string> ym </string> </item>
  </set2>
</instance>
```

(For technical reasons, we enclose every string x_i and y_j by a `string`-element *and* an `item`-element. For the proof of Theorem 12, one of the two would suffice, but for the proof of Theorem 13 it is more convenient if each x_i and y_j is enclosed by two element nodes.)

It should be clear that, given as input $x_1\#\cdots\#x_m\#y_1\cdots\#y_m\#$, the above XML document can be produced by using a constant number of sequential scans, constant internal memory space, and two external memory tapes.

Proof of Theorem 12:
The SET-EQUALITY problem can be expressed by the following XQuery query $Q :=$

```
<result>
  if ( every $x in /instance/set1/item/string satisfies
         some $y in /instance/set2/item/string satisfies
           $x = $y )
     and
     ( every $y in /instance/set2/item/string satisfies
         some $x in /instance/set1/item/string satisfies
           $x = $y )
  then <true/>
  else ()
</result>
```

Note that if $\{x_1,..,x_m\} = \{y_1,..,y_m\}$, then Q returns the document `<result><true/></result>`, and otherwise Q returns the "empty" document `<result></result>`. Thus, if Q could be evaluated in *LasVegas*-$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, then the SET-EQUALITY problem could be solved in $\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, contradicting Theorem 6. □

Proof of Theorem 13:
The *XPath* query Q of Figure 1 selects all `item`-nodes below `set1` whose string content does *not* occur as the string content of some `item`-node below `set2` (recall the "existential" semantics of *XPath* [20]). In other words: Q selects all (nodes that represent) elements in $X - Y$, for $X := \{x_1,..,x_m\}$ and $Y := \{y_1,..,y_m\}$.

Now assume, *for contradiction*, that the problem of filtering an input XML document stream with the *XPath* query Q (i.e., checking whether at least one document node is selected by Q) belongs to the class co-$\mathrm{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. Then, clearly, there exists an $\big(o(\log N), O(\sqrt[4]{N}/\log N), O(1)\big)$-bounded randomized Turing machine T which has the following properties for every input $x_1\#\cdots\#x_m\#y_1\#\cdots\#y_m\#$ (where $X := \{x_1,..,x_m\}$ and $Y := \{y_1,..,y_m\}$):

(1) If Q selects at least one node (i.e., $X - Y \neq \emptyset$, i.e., $X \not\subseteq Y$), then T accepts with probability 1.

(2) If Q does not select any node (i.e., $X - Y = \emptyset$, i.e., $X \subseteq Y$), then T rejects with probability ≥ 0.5.

This machine T can be used to solve the SET-EQUALITY problem by a machine $\tilde{T}$ as follows: First, $\tilde{T}$ starts T with input $x_1\#\cdots\#x_m\#y_1\#\cdots\#y_m\#$. Afterwards, $\tilde{T}$ starts T with input $y_1\#\cdots\#y_m\#x_1\#\cdots\#x_m\#$. If both runs reject, then $\tilde{T}$ accepts its entire input. Otherwise, $\tilde{T}$ rejects.
Let us analyze the acceptance/rejectance probabilities of $\tilde{T}$:

(i) If $X \neq Y$, then either $X \not\subseteq Y$ or $Y \not\subseteq X$, and thus, due to (1), at least one of the two runs of T has to accept. The machine $\tilde{T}$ will therefore reject with probability 1.

(ii) If $X = Y$, then $X \subseteq Y$ and $Y \subseteq X$. Due to (2), we therefore know that each of the two runs of T will accept with probability ≥ 0.5 and thus, in total, $\tilde{T}$ will accept with probability ≥ 0.25.

To increase the acceptance probability to 0.5, we can start two independent runs of $\tilde{T}$ and accept if at least one of the two runs accept. In total, this leads to a $\big(o(\log N), O(\sqrt[4]{N}/\log N), O(1)\big)$-bounded

```
descendant::set1 / child::item [ not  child::string  =
                                    ancestor::instance / child::set2 / child::item / child::string ]
```

Figure 1: The *XPath* query Q used in the proof of Theorem 13.

randomized Turing machine which, on every input $x_1\#\cdots\#x_m\#$ $y_1\#\cdots\#y_m\#$,

– accepts with probability ≥ 0.5, if $\{x_1,..,x_m\} = \{y_1,..,y_m\}$,

– rejects with probability 1, otherwise.

In other words: This machine shows that the SET-EQUALITY problem belongs to $\text{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, contradicting Theorem 6. Therefore, the problem of filtering an input XML document stream with the *XPath* query Q does not belong to the class co-$\text{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. □

5. LIST MACHINES

This section as well as the subsequent sections are devoted to the proof of Theorem 6. For proving Theorem 6 we use *list machines*. The important advantage that these list machines have over the original Turing machines is that they make it fairly easy to track the "flow of information" during a computation.

In [10] we introduced the notion of deterministic list machines with output. In what follows, we propose a nondeterministic version of such machines without output, i.e., nondeterministic list machines for solving decision problems. To introduce *nondeterminism* to the notion of [10] requires some care — the straightforward approach where, instead of the transition functions used in [10], transition *relations* are allowed, will lead to a machine model that is too weak for adequately simulating nondeterministic Turing machines. Therefore, instead of using transition *relations*, the following notion of nondeterministic list machines allows explicit nondeterministic choices in transitions.

Definition 14 (Nondeterministic List Machine).
A *nondeterministic list machine (NLM)* is a tuple

$$M \quad = \quad (t, m, I, C, A, a_0, \alpha, B, B_{acc})$$

consisting of

- a $t \in \mathbb{N}$, the *number of lists*.
- an $m \in \mathbb{N}$, the *length of the input*.
- a finite set I whose elements are called *input numbers* (usually, $I \subseteq \mathbb{N}$ or $I \subseteq \{0,1\}^*$).
- a finite set C whose elements are called *nondeterministic choices*.
- a finite set A whose elements are called *(abstract) states*.
  We assume that I, C, and A are pairwise disjoint and do not contain the two special symbols '⟨' and '⟩'.
  We call $\mathbb{A} := I \cup C \cup A \cup \{\langle, \rangle\}$ the *alphabet* of the machine.
- an *initial state* $a_0 \in A$.
- a *transition function*

$$\alpha \;:\; (A \setminus B) \times \left(\mathbb{A}^*\right)^t \times C \;\to\; \left(A \times \text{Movement}^t\right)$$

  with $\text{Movement} := \{ (\textit{head-direction}, \textit{move}) \mid \textit{head-direction} \in \{-1,+1\}, \textit{move} \in \{\textit{true}, \textit{false}\} \}$.
- a set $B \subseteq A$ of *final states*.
- a set $B_{acc} \subseteq B$ of *accepting states*. (We use $B_{rej} := B \setminus B_{acc}$ to denote the set of *rejecting* states.) ⊣

Intuitively, an NLM $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ operates as follows: The input is a sequence $(v_1, \ldots, v_m) \in I^m$. Instead of tapes (as a Turing machine), an NLM operates on t lists. In particular, this means that a new list cell can be inserted between two existing cells. As for tapes, there is a read-write head operating on each list. Cells of the lists store strings in $\mathbb{A}^*$ (and not just symbols from $\mathbb{A}$). Initially, the first list, called the *input list*, contains $(v_1, \ldots, v_m)$, and all other lists are empty. The heads are on the left end of the lists. The transition function only determines the NLM's new state and the head movements, and *not what is written into the list cells*. In each step of the computation, the heads move according to the transition function, by choosing "nondeterministically" an arbitrary element in C. In each computation step, the current state, the content of all current head positions, and the nondeterministic choice $c \in C$ used in the current transition, are written *behind* each head. When a final state is reached, the machine stops. If this final state belongs to B_{acc}, the according run is *accepting*; otherwise it is *rejecting*. Figure 2 illustrates a transition of an NLM. The formal definition of the semantics of nondeterministic list machines can be found in Appendix B.

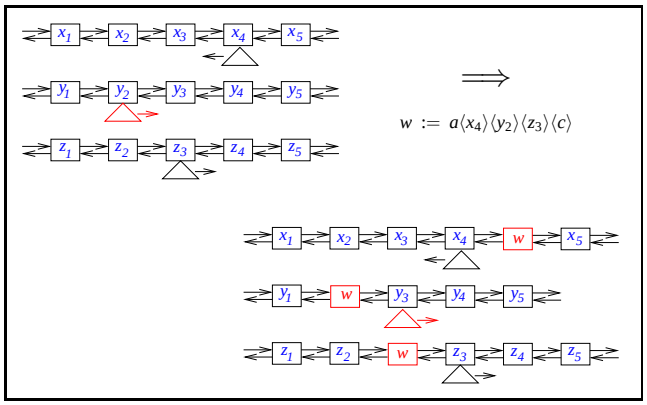

Figure 2: A transition of an NLM. The example transition is of the form $(a, x_4, y_2, z_3, c) \to (b, (-1, \textbf{\textit{false}}), (1, \textbf{\textit{true}}), (1, \textbf{\textit{false}}))$. The new string w that is written in the tape cells consists of the current state a, the content of the list cells read before the transition, and the nondeterministic choice c.

An NLM is called *deterministic* if $|C| = 1$.
For every run ρ of an NLM M and for each list τ of M, we define $\text{rev}(\rho, \tau)$ to be the number of changes of the direction of the τ-th list's head in run ρ. We say that M is (r,t)-*bounded*, for some $r, t \in \mathbb{N}$, if it has at most t lists, every run ρ of M is finite, and $1 + \sum_{\tau=1}^{t} \text{rev}(\rho, \tau) \;\leq\; r$.

Randomized list machines are defined in a similar way as randomized Turing machines: For configurations γ and γ' of an NLM M, the probability $\Pr(\gamma \to_M \gamma')$ that γ yields γ' in one step, is defined as $|\{c \in C : \gamma' \text{ is the } c\text{-successor of } \gamma\}|/|C|$. For a run $\rho = (\rho_1, .., \rho_\ell)$, the probability $\Pr(\rho)$ that M performs run ρ is the product of the probabilities $\Pr(\rho_i \to_M \rho_{i+1})$, for all $i < \ell$. For an input $\overline{v} \in I^m$, the probability that M accepts $\overline{v}$ is defined as the sum of $\Pr(\rho)$ for all accepting runs ρ of M on input $\overline{v}$.

The following notation will be very convenient:

Definition 15 ($\rho_M(\overline{v},\overline{c})$). Let M be an NLM and let $\ell \in \mathbb{N}$ such that every run of M has length $\leq \ell$. For every input $\overline{v} \in I^m$ and every sequence $\overline{c} = (c_1,..,c_\ell) \in C^\ell$, we use $\rho_M(\overline{v},\overline{c})$ to denote the run $(\rho_1,..,\rho_k)$ obtained by starting M with input $\overline{v}$ and by making in its i-th step the nondeterministic choice c_i (i.e., ρ_{i+1} is the c_i-successor of ρ_i). ⊣

6. LIST MACHINES CAN SIMULATE TURING MACHINES

An important property of list machines is that they can simulate Turing machines in the following sense:

Lemma 16 (Simulation Lemma). *Let $r,s : \mathbb{N} \to \mathbb{N}$, $t \in \mathbb{N}$, and let $T = (Q,\Sigma,\Delta,q_0,F,F_{acc})$ be an (r,s,t)-bounded NTM with a total number of $t+u$ tapes and with $\{\Box,\#\} \subseteq \Sigma$. Then for every $m,n \in \mathbb{N}$ there exists an $(r(m\cdot(n+1)),t)$-bounded NLM $M_{m,n} = M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ with $I = \big(\Sigma\setminus\{\Box,\#\}\big)^n$ and $|C| \leq 2^{O(\ell(m\cdot(n+1)))}$, where $\ell(N)$ is an upper bound on the length of T's runs on input words of length N, and*

$$|A| \;\leq\; 2^{d\cdot t^2\cdot r(m\cdot(n+1))\cdot s(m\cdot(n+1)) + 3t\cdot\log(m\cdot(n+1))}, \qquad (2)$$

for some number $d = d(u,|Q|,|\Sigma|)$ that does not *depend on r, m, n, t, such that for all $\overline{v} = (v_1,..,v_m) \in I^m$ we have*

$$\Pr(M \text{ accepts } \overline{v}) \;=\; \Pr(T \text{ accepts } v_1\#\cdots v_m\#)\,.$$

Furthermore, if T is deterministic, then M is deterministic, too. ⊣

In Section 8 we will use the simulation lemma to transfer the lower bound results for list machines to lower bound results for Turing machines.

In [10], the simulation lemma has been stated and proved for deterministic machines. For nondeterministic machines, the construction is based on the same idea. However, some further work is necessary to assure that the according list machine accepts with the same probability as the given Turing machine. Throughout the remainder of this section, the proof idea is given; a detailed proof of Lemma 16 can be found in Appendix C. For proving Lemma 16, the following straightforward characterization of probabilities for Turing machines is very convenient.

Definition 17 (C_T and $\rho_T(w,\overline{c})$). Let T be an NTM for which there exists a function $\ell : \mathbb{N} \to \mathbb{N}$ such that every run of T on a length N input word has length at most $\ell(N)$. Let $b := \max\{|\mathrm{Next}_T(\gamma)| : \gamma \text{ is a configuration of } T\}$ be the maximum branching degree of T (note that b is finite since T's transition relation is finite). Let $b' := \mathrm{lcm}\{1,..,b\}$ be the least common multiple of the numbers 1,2,..,b, and let $C_T := \{1,..,b'\}$. For every $N \in \mathbb{N}$, every input word $w \in \Sigma^*$ of length N, and every sequence $\overline{c} = (c_1,\cdots,c_{\ell(N)}) \in (C_T)^{\ell(N)}$, we define $\rho_T(w,\overline{c})$ to be the run $(\rho_1,..,\rho_k)$ of T that is obtained by starting T with input w and by choosing in its i-th computation step the $\big(c_i \bmod |\mathrm{Next}_T(\rho_i)|\big)$-th of the $|\mathrm{Next}_T(\rho_i)|$ possible next configurations. ⊣

Lemma 18. *Let T be an NTM for which there exists a function $\ell : \mathbb{N} \to \mathbb{N}$ such that every run of T on a length N input word has length at most $\ell(N)$, and let C_T be chosen according to Definition 17. Then we have for every run ρ of T on an input w of length N that*

$$\Pr(\rho) \;=\; \frac{|\{\overline{c}\in(C_T)^{\ell(N)} : \rho_T(w,\overline{c})=\rho\}|}{|(C_T)^{\ell(N)}|}, \quad \textit{and, in total,}$$

$$\Pr(T \textit{ accepts } w) \;=\; \frac{|\{\overline{c}\in(C_T)^{\ell(N)} : \rho_T(w,\overline{c}) \textit{ accepts}\}|}{|(C_T)^{\ell(N)}|}\,.$$

The (straightforward) proof can be found in Appendix A.

For proving Lemma 16, let T be an NTM. We construct an NLM M that simulates T. The lists of M represent the external memory tapes of T. More precisely, the cells of the lists of M represent segments, or *blocks*, of the corresponding external memory tapes of T in such a way that the content of a block at any step of the computation can be reconstructed from the content of the cell representing it. The blocks evolve dynamically in a way that is described below. M's set C of *nondeterministic choices* is defined as $C := (C_T)^\ell$, where C_T is chosen according to Definition 17 and $\ell := \ell(m\cdot(n+1))$ is an upper bound on T's running time and tape length, obtained from Lemma 3. Each step of the list machine corresponds to the sequence of Turing machine steps that are performed by T while none of its external memory tape heads changes its direction or leaves its current tape block. Of course, the length ℓ' of this sequence of T's steps is bounded by T's entire running time ℓ. Thus, if $c = (c_1,..,c_\ell) \in C = (C_T)^\ell$ is the nondeterministic choice used in M's current step, the prefix of length ℓ' of c tells us, which nondeterministic choices (in the sense of Definition 17) T makes throughout the corresponding sequence of ℓ' steps. The *states* of M encode:

- The current state of the Turing machine T.
- The content and the head positions of the internal memory tapes $t+1,\ldots,t+u$ of T.
- The head positions of the external memory tapes $1,\ldots,t$.
- For each of the external memory tapes $1,\ldots,t$, the boundaries of the block in which the head currently is.

Representing T's current state and the content and head positions of the u internal memory tapes requires $|Q|\cdot 2^{O(s(m\cdot(n+1)))}\cdot s(m\cdot(n+1))^u$ states. The t head positions of the external memory tapes increase the number of states by a factor of ℓ^t. The $2t$ block boundaries increase the number of states by another factor of ℓ^{2t}. So overall, the number of states is bounded by $|Q|\cdot 2^{O(s(m\cdot(n+1)))}\cdot s(m\cdot(n+1))^u\cdot\ell^{3t}$. By Lemma 3, this yields the bound (2).

Initially, for an input word $v_1\#\cdots v_m\#$, the first Turing machine tape is split into m blocks which contain the input segments $v_i\#$ (for $1 \leq i < m$), respectively, $v_m\#\Box^{\ell-(n+1)}$ (that is, the m-th input segment is padded by as many blank symbols as the Turing machine may enter throughout its computation). All other tapes just consist of one block which contains the blank string $\Box^\ell$. The heads in the initial configuration of M are on the first cells of their lists. Now we start the simulation: For a particular nondeterministic choice $c_1 = (c_{11},c_{12},..,c_{1\ell}) \in C = (C_T)^\ell$, we start T's run $\rho_T(v_1\#\cdots,c_{11}c_{12}c_{13}\cdots)$. As long as no head of the external memory tapes of T changes its direction or crosses the boundaries of its current block, M does not do anything. If a head on a tape $i_0 \in \{1,\ldots,t\}$ crosses the boundaries of its block, the head i_0 of M moves to the next cell, and the previous cell is overwritten with sufficient information so that if it is visited again later, the content of the corresponding block of tape i_0 of T can be reconstructed. The blocks on all other tapes are split behind the current head position ("behind" is defined relative to the current direction in which the head moves). A new cell is inserted into the lists behind the head, this cell represents the newly created tape block that is behind the head. The newly created block starting with the current head position is represented by the (old) cell on which the head still stands. The case that a head on a tape $i_0 \in \{1,\ldots,t\}$ changes its direction is treated similarly.

The simulation stops as soon as T has reached a final state; and M accepts if, and only if, T does. A close look at the possible runs of T and M shows that M has the same acceptance probabilities as T.

7. LOWER BOUNDS FOR LIST MACHINES

This section's main result is that it provides constraints on a list machine's parameters, which ensure that list machines which comply to these constraints can neither solve the *multiset equality problem* nor the *checksort problem*. In fact, we can show a slightly stronger result, the precise formulation of which requires the following observation.

Definition 19 (sortedness). Let $m \in \mathbb{N}$ and let π be a permutation of $\{1,..,m\}$. We define $sortedness(\pi)$ to be the length of the longest subsequence[2] of $\big(\pi(1),..,\pi(m)\big)$ that is sorted in either ascending or descending order (i.e., that is a subsequence of $(1,..,m)$ or of $(m,..,1)$). ⊣

Remark 20. It is well-known that for every permutation π of $\{1,..,m\}$, $sortedness(\pi) \in \Omega(\sqrt{m})$, and that there exists a particular permutation $\varphi := \varphi_m$ with $sortedness(\varphi) \leq 2\sqrt{m}-1$. In fact, one way of finding such a permutation is to let $\big(\varphi(1),..,\varphi(m)\big)$ be the numbers $1,..,m$, sorted lexicographically by their reverse binary representation. ⊣

Lemma 21 (Lower Bound for List Machines).
Let $k,m,n,r,t \in \mathbb{N}$ such that m is a power of 2 and $t \geq 2$, $m \geq 24\cdot(t+1)^{4r}+1$, $k \geq 2m+3$, $n \geq 1+(m^2+1)\cdot\log(2k)$. We let $I := \{0,1\}^n$, identify I with the set $\{0,1,..,2^n-1\}$, and divide it into m consecutive intervals $I_1,..,I_m$, each of length $2^n/m$. Let φ be a permutation of $\{1,..,m\}$ with $sortedness(\varphi) \leq 2\sqrt{m}-1$, and let $\mathscr{I} := I_{\varphi(1)} \times \cdots \times I_{\varphi(m)} \times I_1 \times \cdots \times I_m$.
Then there is no (r,t)-bounded NLM $M = (t,2m,I,C,A,a_0,\alpha,B,B_{acc})$ with $|A| \leq k$ and $I = \{0,1\}^n$, such that for all $\overline{v} = (v_1,..,v_m,v'_1,..,v'_m) \in \mathscr{I}$ we have:
If $(v_1,..,v_m) = (v'_{\varphi(1)},..,v'_{\varphi(m)})$, then $\Pr(M \text{ accepts } \overline{v}) \geq \frac{1}{2}$; otherwise $\Pr(M \text{ accepts } \overline{v}) = 0$. ⊣

It is straightforward to see that the above lemma, in particular, implies that neither the *(multi)set equality problem* nor the *checksort problem* can be solved by list machines with the according parameters.

The proof of Lemma 21 is based on the following ideas (the detailed proof is given in Appendix D):

1. Suppose for contradiction that M is an NLM that meets the lemma's requirements.

2. Observe that there exists an upper bound ℓ on the length of M's runs (Lemma 31 (a) in Appendix D) and a particular sequence $\overline{c} = (c_1,..,c_\ell) \in C^\ell$ of nondeterministic choices (Lemma 26), such that for at least half of the inputs $\overline{v} := (v_1,..,v_m,v'_1,..,v'_m) \in \mathscr{I}$ with $(v_1,..,v_m) = (v'_{\varphi(1)},..,v'_{\varphi(m)})$, the particular run $\rho_M(\overline{v},\overline{c})$ accepts.
   We let $\mathscr{I}_{acc,\overline{c}} := \{\overline{v} \in \mathscr{I} : \rho_M(\overline{v},\overline{c}) \text{ accepts}\}$ and, from now on, we only consider runs that are generated by the fixed sequence $\overline{c}$ of nondeterministic choices.

3. Show that, throughout its computation, M can "mix" the relative order of its input values only to a rather limited extent (cf., Lemma 37). This can be used to show that for every run of M on every input $(v_1,..,v_m,v'_1,..,v'_m) \in \mathscr{I}$ there must be an index i_0 such that v_{i_0} and $v'_{\varphi(i_0)}$ are never compared throughout this run.

4. Use the notion of the *skeleton* of a run (cf., Definition 28), which, roughly speaking, is obtained from a run by replacing every input value v_i with its index i and by replacing every nondeterministic choice $c \in C$ with the wildcard symbol "?". In particular, the skeleton contains input *positions* rather than concrete input *values*; but given the skeleton together with the concrete input values and the sequence of nondeterministic choices, the entire run of M can be reconstructed.

5. Now choose ζ to be a skeleton that is generated by the run $\rho_M(\overline{v},\overline{c})$ for as many input instances $\overline{v} \in \mathscr{I}_{acc,\overline{c}}$ as possible, and use $\mathscr{I}_{acc,\overline{c},\zeta}$ to denote the set of all those input instances.

6. Due to 3. there must be an index i_0 such that for all inputs from $\mathscr{I}_{acc,\overline{c},\zeta}$, the values v_{i_0} and $v'_{\varphi(i_0)}$ (i.e., the values from the input positions i_0 and $m+\varphi(i_0)$) are never compared throughout the run that has skeleton ζ. To simplify notation let us henceforth assume without loss of generality that $i_0 = 1$.

7. Now fix $(v_2,..,v_m)$ such that the number of v_1 with $V(v_1) := (v_1,v_2,..,v_m,v_{\varphi^{-1}(1)},..,v_{\varphi^{-1}(m)}) \in \mathscr{I}_{acc,\overline{c},\zeta}$ is as large as possible.

8. Argue that, for our fixed $(v_2,..,v_m)$, there must be at least two distinct v_1 and w_1 such that $V(v_1) \in \mathscr{I}_{acc,\overline{c},\zeta}$ and $V(w_1) \in \mathscr{I}_{acc,\overline{c},\zeta}$. This is achieved by observing that the number of skeletons depends on the machine's parameters t, r, m, k, but *not* on n (Lemma 32) and by using the lemma's assumption on the machine's parameters t,r,m,k,n.

9. Now we know that the input values of $V(v_1)$ and $V(w_1)$ coincide on all input positions except 1 and $m+\varphi(1)$. From 3. we know that the values from the positions 1 and $m+\varphi(1)$ are never compared throughout M's (*accepting*) runs $\rho_M\big(V(v_1),\overline{c}\big)$ and $\rho_M\big(V(w_1),\overline{c}\big)$. From this we obtain (cf., Lemma 34) an accepting run $\rho_M(\overline{u},\overline{c})$ of M on input

$$\begin{aligned}\overline{u} &:= (u_1,..,u_m,u'_1,..,u'_m)\\ &:= \big(v_1,v_2,..,v_m,w_{\varphi^{-1}(1)},w_{\varphi^{-1}(2)},..,w_{\varphi^{-1}(m)}\big).\end{aligned}$$

   In particular, this implies that $\Pr(M \text{ accepts } \overline{u}) > 0$. However, for this particular input $\overline{u}$ we know that $u_1 = v_1 \neq w_1 = u_{\varphi(1)}$, and therefore, $(u_1,..,u_m) \neq (u'_{\varphi(1)},..,u'_{\varphi(m)})$. This gives us a contradiction to the assumption that $\Pr(M \text{ accepts } \overline{v}) = 0$ for all inputs $\overline{v} = (v_1,..,v_m,v'_1,..,v'_m)$ with $(v_1,..,v_m) \neq (v'_{\varphi(1)},..,v'_{\varphi(m)})$.

8. LOWER BOUNDS FOR TURING MACHINES

Lemma 22. *Let $r,s : \mathbb{N} \to \mathbb{N}$ such that $r(N) = o(\log N)$ and $s(N) = o\big(\sqrt[4]{N}/r(N)\big)$. Then, there is no $(r,s,O(1))$-bounded $(\frac{1}{2},0)$-RTM that solves the following problem* CHECK-φ. ⊣

CHECK-φ
Instance: $v_1\#\ldots v_m\#v'_1\#\ldots v'_m\#$, where $m \geq 0$ is a power of 2, and $(v_1,\ldots,v_m,v'_1,\ldots,v'_m) \in I_{\varphi(1)} \times \cdots \times I_{\varphi(m)} \times I_1 \times \cdots \times I_m$, where $\varphi := \varphi_m$ is the permutation of $\{1,..,m\}$ obtained from Remark 20, and the sets $I_1,..,I_m \subseteq \{0,1\}^{m^3}$ are obtained as the partition of the set $\{0,1\}^{m^3}$ into m consecutive subsets, each of size $2^{(m^3)}/m$.
Problem: Decide if $(v_1,\ldots,v_m) = (v'_{\varphi(1)},\ldots,v'_{\varphi(m)})$.

[2] A sequence $(s_1,..,s_\lambda)$ is a *subsequence* of a sequence $(s'_1,..,s'_{\lambda'})$, if there exist indices $j_1 < \cdots < j_\lambda$ such that $s_1 = s'_{j_1}$, $s_2 = s'_{j_2}$, ..., $s_\lambda = s'_{j_\lambda}$.

Proof: Suppose for contradiction that there is a $t \in \mathbb{N}$ such that the problem CHECK-φ is solved by an (r,s,t)-bounded $(\frac{1}{2},0)$-RTM T. Without loss of generality we may assume that $t \geq 2$.

Let d be the constant introduced in Lemma 16 (the simulation lemma). Let m be a sufficiently large power of 2 such that

$$m \geq 24 \cdot (t+1)^{4 \cdot r(2m \cdot (m^3+1))} + 1 \quad \text{and} \tag{3}$$

$$m^3 \geq 1 + d \cdot t^2 \cdot r(2m \cdot (m^3+1)) \cdot s(2m \cdot (m^3+1)) + 3t \cdot \log(2m \cdot (m^3+1)). \tag{4}$$

Such an m exists because $r(N) = o(\log N)$ (for Equation (3)) and $r(N) \cdot s(N) = o\left(\sqrt[4]{N}\right)$ (for Equation (4)). Let $n := m^3$ and $I := \{0,1\}^n$. By Lemma 16, there is an $\big(r(2m \cdot (n+1)), t\big)$-bounded NLM $M = (t, 2m, I, C, A, a_0, \alpha, B, B_{acc})$ with

$$k \leq 2^{d \cdot t^2 \cdot r(2m \cdot (n+1)) \cdot s(2m \cdot (n+1)) + 3t \cdot \log(2m \cdot (n+1))}$$

states that simulates T on inputs from I^{2m}. In particular, for all instances

$$\overline{v} = (v_1, .., v_m, v'_1, .., v'_m) \in I_{\varphi(1)} \times \cdots \times I_{\varphi(m)} \times I_1 \times \cdots \times I_m$$

the following is true:

$(*)$: If $(v_1, .., v_m) = (v'_{\varphi(1)}, .., v'_{\varphi(m)})$, then $\Pr(M \text{ accepts } \overline{v}) \geq \frac{1}{2}$; otherwise $\Pr(M \text{ accepts } \overline{v}) = 0$.

We clearly have $k \geq 2m+3$. By (3), we have

$$m \geq 24 \cdot (t+1)^{4 \cdot r(2m \cdot (m^3+1))} + 1 = 24 \cdot (t+1)^{4 \cdot r(2m \cdot (n+1))} + 1.$$

By (4), we have

$$\begin{aligned} n &= m^3 \\ &\geq 1 + (m^2+1) \cdot \big(1 + d \cdot t^2 \cdot r(2m \cdot (m^3+1)) \cdot s(2m \cdot (m^3+1)) + 3t \cdot \log(2m \cdot (m^3+1))\big) \\ &= 1 + (m^2+1) \cdot \big(1 + d \cdot t^2 \cdot r(2m \cdot (n+1)) \cdot s(2m \cdot (n+1)) + 3t \cdot \log(2m \cdot (n+1))\big) \\ &\geq 1 + (m^2+1) \cdot \big(\log(2) + \log(k)\big) \\ &= 1 + (m^2+1) \cdot \log(2k). \end{aligned}$$

Thus, $(*)$ is a contradiction to Lemma 21, and the proof of Lemma 22 is complete. □

Proof of Theorem 6: For inputs that are instances of the problem CHECK-φ, the problems SET-EQUALITY, MULTISET-EQUALITY, CHECK-SORT, and CHECK-φ coincide. Thus, Lemma 22 immediately implies Theorem 6. □

9. CONCLUSION

We have proved tight lower bounds for the natural decision problems *(multi)set equality* and *checksort*, in our Turing machine based computation model for processing large data sets. These lower bounds do not only hold for deterministic, but even for *randomized* algorithms with one-sided bounded error probability. Our results are obtained by carefully analyzing the flow of information in a Turing machine computation.

As applications of these lower bound results, we obtained lower bounds on the worst case data complexity of query evaluation for the languages *XQuery*, *XPath*, and *relational algebra* on data streams.

We complement our lower bounds for *checksort* and *(multi)set equality* by proving that these problems can be solved by nondeterministic machines and by randomized machines with complementary one-sided error probabilities. As a consequence, we obtain a separation between the deterministic, the randomized, the co-randomized, and the nondeterministic external memory complexity classes.

A specific problem for which we could not prove lower bounds, even though it looks very similar to the set equality problem, is the *disjoint sets problem*, which asks if two given sets of strings are disjoint. Another important future task is to develop techniques for proving lower bounds (a) for randomized computations with two-sided bounded error and (b) for appropriate problems in a setting where $\Omega(\log N)$ head reversals (i.e., sequential scans of external memory devices) are available.

10. REFERENCES

[1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proc. FOCS'04*, pages 540–549, 2004.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

[3] L. Arge and P. Bro Miltersen. On showing lower bounds for external-memory computational geometry problems. In J. Abello and J. Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 139–159. 1999.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. PODS'02*, pages 1–16, 2002.

[5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proc. PODS'04*, pages 177–188, 2004.

[6] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proc. PODS'05*, pages 216–227, 2005.

[7] J. Chen and C.-K. Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.

[8] M. Grohe, C. Koch, and N. Schweikardt. The complexity of querying external memory and streaming data. In *Proc. FCT'05*, volume 3623 of *Springer LNCS*, pages 1–16, 2005.

[9] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. In *Proc. ICALP'05*, volume 3580 of *Springer LNCS*, pages 1076–1088, 2005. (Best paper award at ICALP'05, track B.).

[10] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proc. PODS'05*, pages 238–249, 2005.

[11] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series In Discrete Mathematics And Theoretical Computer Science, 1999.

[12] J. Hromkovic. *Design and Analysis of Randomized Algorithms*. Springer-Verlag, 1998.

[13] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *Springer LNCS*. 2003.

[14] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[15] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[16] S. Muthukrishnan. Data streams: algorithms and applications. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 413–413, 2003.

[17] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[18] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.

[19] K. Wagner and G. Wechsung. *Computational Complexity*. VEB Deutscher Verlag der Wissenschaften, 1986.

[20] World Wide Web Consortium. XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005, 2005. `http://www.w3.org/TR/xpath20/`.

APPENDIX

A. TURING MACHINE BASICS

Definition 23 (Notation concerning Turing machines).
Let $T = (Q, \Sigma, \Delta, q_0, F, F_{acc})$ be a nondeterministic Turing machine (NTM, for short) with $t+u$ tapes, where Q is the state space, Σ the alphabet, $q_0 \in Q$ the start state, $F \subseteq Q$ the set of final states, $F_{acc} \subseteq F$ the set of *accepting* states, and

$$\Delta \ \subseteq \ (Q \setminus F) \times \Sigma^{t+u} \times Q \times \Sigma^{t+u} \times \{L, N, R\}^{t+u}$$

the transition relation. Here L, N, R are special symbols indicating the head movements.

We assume that all tapes are one-sided infinite and have cells numbered $1, 2, 3,$ etc, and that $\square \in \Sigma$ is the "blank" symbol which, at the beginning of the TM's computation, is the inscription of all empty tape cells.

A *configuration* of T is a tuple

$$(q, p_1, \ldots, p_{t+u}, w_1, \ldots, w_{t+u}) \ \in \ Q \times \mathbb{N}^{t+u} \times (\Sigma^*)^{t+u},$$

where q is the current state, $p_1, \ldots, p_{t+u}$ are the positions of the heads on the tapes, and $w_1, \ldots, w_{t+u}$ are the contents of the tapes. For a configuration γ we write $\mathrm{Next}_T(\gamma)$ for the set of all configurations γ' that can be reached from γ in a single computation step.

A configuration is called *final* (resp., *accepting*) if its current state q is final (resp., accepting), that is, $q \in F$ (resp., $q \in F_{acc}$). Note that a final configuration does not have a successor configuration.

A *run* of T is a sequence $\rho = (\rho_j)_{j \in J}$ of configurations ρ_j satisfying the obvious requirements. We are only interested in finite runs here, where the index set J is $\{1, \ldots, \ell\}$ for an $\ell \in \mathbb{N}$, and where ρ_ℓ is *final*.

When considering *decision problems*, a run ρ is called *accepting* (resp., *rejecting*) if its final configuration is accepting (resp., rejecting). When considering, instead, Turing machines that produce an output, we say that a run ρ *outputs the word* w' if ρ end in an *accepting* state and w' is the inscription of the last (i.e., t-th) external memory tape. If ρ ends in a *rejecting* state, we say that ρ *outputs "I don't know"*.

Without loss of generality we assume that our Turing machines are *normalized* in such a way that in each step at most one of its heads moves to the left or to the right. ⊣

Proof of Lemma 18:
To prove (a), observe that for every run $\rho = (\rho_1, .., \rho_k)$ of T on w we have $k \leq \ell(N)$, and

$$\frac{|\{\overline{c} \in C_T^{\ell(N)} : \rho_T(w,\overline{c}) = \rho\}|}{|C_T^{\ell(N)}|} \ = \ \frac{1}{|C_T^{\ell(N)}|} \cdot \left(\prod_{i=1}^{k-1} \frac{|C_T|}{|\mathrm{Next}_T(\rho_i)|} \right) \cdot |C_T|^{\ell(N)-(k-1)} \ = \ \prod_{i=1}^{k-1} \frac{1}{|\mathrm{Next}_T(\rho_i)|} \ = \ \Pr(\rho) \ .$$

(b) follows directly from (a), since

$$\Pr(T \text{ accepts } w) \ \stackrel{\text{def}}{=} \ \sum_{\substack{\rho \ : \ \rho \text{ is an accepting} \\ \text{run of } T \text{ on } w}} \Pr(\rho) \ \stackrel{\text{(a)}}{=} \ \sum_{\substack{\rho \ : \ \rho \text{ is an accepting} \\ \text{run of } T \text{ on } w}} \frac{|\{\overline{c} \in C_T^{\ell(N)} : \rho_T(w,\overline{c}) = \rho\}|}{|C_T^{\ell(N)}|}$$

$$= \ \sum_{\substack{\overline{c} \in C_T^{\ell(N)} \ : \\ \rho_T(w,\overline{c}) \text{ accepts}}} \frac{1}{|C_T^{\ell(N)}|} \ = \ \frac{|\{\overline{c} \in C_T^{\ell(N)} : \rho_T(w,\overline{c}) \text{ accepts}\}|}{|C_T^{\ell(N)}|} \ . \qquad \square$$

B. FORMAL DEFINITION OF THE SEMANTICS OF NONDETERMINISTIC LIST MACHINES

Formally, the semantics of nondeterministic list machines are defined as follows:

Definition 24 (Semantics of NLMs).

(a) A *configuration* of an NLM $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ is a tuple $(a, \overline{p}, \overline{d}, X)$ with

$$\overline{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_t \end{pmatrix} \ \in \ \mathbb{N}^t \ , \quad \overline{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_t \end{pmatrix} \ \in \ \{-1, +1\}^t \ , \quad X = \begin{pmatrix} \overline{x}_1 \\ \vdots \\ \overline{x}_t \end{pmatrix} \ \in \ \left((\mathbb{A}^*)^*\right)^t \ ,$$

where $\mathbb{N} = \{1, 2, 3, ..\}$ is the set of positive integers,

- $a \in A$ is the *current state*,
- $\overline{p}$ is the tuple of *head positions*,

– $\overline{d}$ is the tuple of *head directions*,

– $\overline{x}_i = (x_{i,1},\ldots,x_{i,m_i}) \in (\mathbb{A}^*)^{m_i}$ for some $m_i \geq 1$, contains the *content of the cells*. (The string $x_{i,j} \in \mathbb{A}^*$ is the content of the jth cell of the ith list.)

(b) The *initial configuration* for input $(v_1,\ldots,v_m) \in I^m$ is a tuple $(a,\overline{p},\overline{d},X)$, where $a = a_0$, $\overline{p} = (1,\ldots,1)^\top$, $\overline{d} = (+1,\ldots,+1)^\top$, and $X = (\overline{x}_1,\ldots,\overline{x}_t)^\top$ with

$$\overline{x}_1 \;=\; \big(\langle v_1\rangle,\ldots,\langle v_m\rangle\big) \;\in\; (\mathbb{A}^*)^m$$

and $\overline{x}_2 = \cdots = \overline{x}_t = \big(\langle\rangle\big) \in (\mathbb{A}^*)^1$.

(c) For a nondeterministic choice $c \in C$, the *c-successor* of a configuration $(a,\overline{p},\overline{d},X)$ is the configuration $(a',\overline{p}',\overline{d}',X')$ defined as follows: Suppose that

$$\alpha\big(a, x_{1,p_1},..,x_{t,p_t}, c\big) \;=\; \big(b, e_1,\ldots,e_t\big).$$

We let $a' = b$. For $1 \leq i \leq t$, let m_i be the length of the list $\overline{x}_i$, and let

$$e'_i \;:=\; (\textit{head-direction}_i, \textit{move}_i) \;:=\; \begin{cases} (-1,\textit{false}) & \text{if } p_i = 1 \text{ and } e_i = (-1,\textit{true}),\\ (+1,\textit{false}) & \text{if } p_i = m_i \text{ and } e_i = (+1,\textit{true}),\\ e_i & \text{otherwise.}\end{cases}$$

This will prevent the machine from "falling off" the left or right end of a list. I.e., if the head is standing on the rightmost (resp., leftmost) list cell, it will stay there instead of moving a further step to the right (resp., to the left).
We fix $f_i \in \{0,1\}$ such that $f_i = 1$ iff $\big(\textit{move}_i = \textit{true}$ or $\textit{head-direction}_i \neq d_i\big)$.
If $f_i = 0$ for *all* $i \in \{1,..,t\}$, then we let $\overline{p}' := \overline{p}$, $\overline{d}' := \overline{d}$, and $X' := X$ (i.e., if none of the machine's head moves, then the *state* is the only thing that may change in the machine's current step).
So suppose that there is at least one i such that $f_i \neq 0$. In this case, we let

$$y \quad := \quad a\,\langle x_{1,p_1}\rangle\cdots\langle x_{t,p_t}\rangle\,\langle c\rangle\,.$$

For all $i \in \{1,..,t\}$, we let

$$\overline{x}'_i \;:=\; \begin{cases} \big(x_{i,1},\ldots,x_{i,p_i-1},y,x_{i,p_i+1},\ldots,x_{i,m_i}\big) & \text{if } \textit{move}_i = \textit{true},\\ \big(x_{i,1},\ldots,x_{i,p_i-1},y,x_{i,p_i},x_{i,p_i+1},\ldots,x_{i,m_i}\big) & \text{if } d_i = +1 \text{ and } \textit{move}_i = \textit{false},\\ \big(x_{i,1},\ldots,x_{i,p_i-1},x_{i,p_i},y,x_{i,p_i+1},\ldots,x_{i,m_i}\big) & \text{if } d_i = -1 \text{ and } \textit{move}_i = \textit{false},\end{cases}$$

and, finally,

$$p'_i \;:=\; \begin{cases} p_i+1 & \text{if } e'_i = (+1,\textit{true}),\\ p_i-1 & \text{if } e'_i = (-1,\textit{true}),\\ p_i+1 & \text{if } e'_i = (+1,\textit{false}),\\ p_i & \text{if } e'_i = (-1,\textit{false}).\end{cases}$$

(d) A configuration $(a,\overline{p},\overline{d},X)$ is *final* (*accepting*, resp., *rejecting*), if $a \in B$ (B_{acc}, resp., $B_{rej} := B \setminus B_{acc}$).
A (finite) *run* of the machine is a sequence $(\rho_1,\ldots,\rho_\ell)$ of configurations, where ρ_1 is the initial configuration for some input, ρ_ℓ is final, and for every $i < \ell$ there is a nondeterministic choice $c_i \in C$ such that ρ_{i+1} is the c_i-successor of ρ_i.
A run is called *accepting* (resp., *rejecting*) if its final configuration is accepting (resp., rejecting).

(e) An input $(v_1,..,v_m) \in I^m$ is *accepted* by machine M if there is at least one accepting run of M on input $(v_1,..,v_m)$. ⊣

It is straightforward to see that

Lemma 25. *Let $M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ be an NLM, and let ℓ be an upper bound on the length of M's runs.*

(a) For every run ρ of M on an input $\overline{v} \in I^m$, we have

$$\Pr(\rho) \;=\; \frac{|\{\overline{c} \in C^\ell : \rho_M(\overline{v},\overline{c}) = \rho\}|}{|C^\ell|}.$$

(b)

$$\Pr(M \textit{ accepts } \overline{v}) \;=\; \frac{|\{\overline{c} \in C^\ell : \rho_M(\overline{v},\overline{c}) \textit{ accepts}\}|}{|C^\ell|}\,.$$

Proof: To prove (a), observe that for every run $\rho = (\rho_1, .., \rho_k)$ of M on $\overline{v}$ we have $k \leq \ell$, and

$$\begin{aligned} \frac{|\{\overline{c} \in C^\ell : \rho_M(\overline{v},\overline{c}) = \rho\}|}{|C^\ell|} &= \frac{1}{|C^\ell|} \cdot \left(\prod_{i=1}^{k-1} |\{c \in C : \rho_{i+1} \text{ is the } c\text{-successor of } \rho_i\}| \right) \cdot |C|^{\ell-(k-1)} \\ &= \prod_{i=1}^{k-1} \Pr(\rho_i \rightarrow_M \rho_{i+1}) \;=\; \Pr(\rho) \;. \end{aligned}$$

(b) follows directly from (a), since

$$\Pr(M \text{ accepts } \overline{v}) \stackrel{\text{def}}{=} \sum_{\substack{\rho \,:\, \rho \text{ is an accepting} \\ \text{run of } M \text{ on } \overline{v}}} \Pr(\rho) \stackrel{\text{(a)}}{=} \sum_{\substack{\rho \,:\, \rho \text{ is an accepting} \\ \text{run of } M \text{ on } \overline{v}}} \frac{|\{\overline{c} \in C^\ell : \rho_M(\overline{v},\overline{c}) = \rho\}|}{|C^\ell|}$$

$$= \sum_{\substack{\overline{c} \in C^\ell \,: \\ \rho_M(\overline{v},\overline{c}) \text{ accepts}}} \frac{1}{|C^\ell|} \quad = \quad \frac{|\{\overline{c} \in C^\ell : \rho(\overline{v},\overline{c}) \text{ accepts}\}|}{|C^\ell|} \;.$$

□

C. PROOF OF THE SIMULATION LEMMA

Let $T = (Q, \Sigma, \Delta, q_0, F, F_{acc})$ be the given (r,s,t)-bounded nondeterministic Turing machine with $t+u$ tapes, where the tapes $1, \ldots, t$ are the external memory tapes and tapes $t+1, \ldots, t+u$ are the internal memory tapes. Let $m, n \in \mathbb{N}$ and $N = m \cdot (n+1)$. Every tuple $\overline{v} = (v_1, .., v_m) \in I^m$ corresponds to an input string $\tilde{v} := v_1 \# v_2 \# \cdots v_m \#$ of length N. Let $r := r(N)$ and $s := s(N)$.

By Lemma 3, there is a constant $c_1 = c_1(u, |Q|, |\Sigma|)$, which does *not* depend on r, m, n, t, such that every run of T on every input $\tilde{v}$, for any $\overline{v} \in I^m$, has length at most

$$\ell(N) \;:=\; N \cdot 2^{c_1 \cdot r \cdot (t+s)} \tag{5}$$

and throughout each such run, each of T's external memory tapes $1, \ldots, t$ has length $\leq \ell(N)$. We let $\ell := \ell(N)$.

Step 1: *Definition of M's set C of nondeterministic choices.*

M's set C of *nondeterministic choices* is chosen as $C := (C_T)^\ell$, where C_T is chosen according to Definition 17. ⊣

Step 2: *Definition of a superset $\tilde{A}$ of M's state set A.*

Let $\hat{Q}$ be the set of potential configurations of tapes $t+1, \ldots, t+u$, together with the current state of T, that is,

$$\hat{Q} := \left\{ (q, p_{t+1}, \ldots, p_{t+u}, w_{t+1}, \ldots, w_{t+u}) \;\middle|\; \begin{array}{l} q \in Q,\ p_{t+i} \in \{1, \ldots, s\}, \\ w_{t+i} \in \Sigma^{\leq s} \text{ (for all } i \in \{1, \ldots, u\}) \end{array} \right\}.$$

Then for a suitable constant $c_2 = c_2(u, |Q|, |\Sigma|)$ we have

$$|\hat{Q}| \;\leq\; 2^{c_2 \cdot s}. \tag{6}$$

We let

$$\begin{aligned} \tilde{A} := \Big\{ (\hat{q},\, \overline{p}_1, .., \overline{p}_t) \;\Big|\; & \hat{q} \in \hat{Q}, \text{ and for each } j \in \{1, .., t\}, \\ & \overline{p}_j = (p_j^{[\![}, p_j^{\uparrow}, p_j^{]\!]}, \textit{head-direction}_j) \text{ with} \\ & \quad p_j^{\uparrow} \in \{1, .., \ell\},\ \textit{head-direction}_j \in \{+1, -1\}, \text{ and} \\ & \quad \text{either } \; p_j^{[\![} = p_j^{]\!]} = \ominus, \\ & \quad \text{or } \; p_j^{[\![}, p_j^{]\!]} \in \{1, .., \ell\} \text{ with } p_j^{[\![} \leq p_j^{\uparrow} \leq p_j^{]\!]} \Big\}. \end{aligned}$$

Here, $\ominus$ is a symbol for indicating that $p_j^{[\![}$ and $p_j^{]\!]}$ are "undefined", that is, that they cannot be interpreted as positions on one of the Turing machine's external memory tapes.

Later, at the end of Step 4, we will specify, *which* particular subset of $\tilde{A}$ will be designated as M's state set A. With *any* choice of A as a subset of $\tilde{A}$ we will have

$$|A| \;\leq\; |\tilde{A}| \;\leq\; |\hat{Q}| \cdot \big(\ell+1\big)^{3 \cdot t} \cdot 2^t \;\leq\; 2^{c_2 \cdot s} \cdot \big(N \cdot 2^{c_1 \cdot r \cdot (t+s)} + 1\big)^{3 \cdot t} \cdot 2^t \;\leq\; 2^{d \cdot t^2 \cdot r \cdot s}$$

for a suitable constant $d = d(u, |Q|, |\Sigma|)$. This completes Step 2. ⊣

Step 3: *Definition of M's initial state a_0 and M's sets B and B_{acc} of final states and accepting states, respectively.*

Let

$$\hat{q}_0 \;:=\; (q_0, \underbrace{1,..,1}_{u}, \underbrace{\Box^s,..,\Box^s}_{u})$$

be the part of T's *initial configuration* that describes the (start) state q_0 of T and the head positions and initial (i.e., empty) content of the tapes $t+1,..,t+u$ (that is, the tapes that represent internal memory).
Let

$$\overline{p}_1 \;:=\; (p_1^{[\![}, p_1^{\uparrow}, p_1^{]\!]}, \textit{head-direction}_1) \;:=\; \begin{cases} (1,1,n+1,+1) & \text{if } m>1 \\ (1,1,\ell,+1) & \text{if } m=0 \end{cases}$$

and, for all $i \in \{2,..,t\}$,

$$\overline{p}_i \;:=\; (p_i^{[\![}, p_i^{\uparrow}, p_i^{]\!]}, \textit{head-direction}_i) \;:=\; (1,1,\ell,+1).$$

As start state of the NLM M we choose

$$a_0 \quad:=\quad (\hat{q}_0, \overline{p}_1, \overline{p}_2, .., \overline{p}_t)$$

As M's sets of *final*, resp., *accepting* states we choose $B := \tilde{B} \cap A$, resp., $B_{acc} := \tilde{B}_{acc} \cap A$ with

$$\begin{aligned} \tilde{B} \quad &:= \quad \big\{ (\hat{q}, \overline{p}_1, \overline{p}_2, .., \overline{p}_t) \in \tilde{A} \;\big|\; \hat{q} \text{ is of the form } (q, \overline{p}, \overline{y}) \in \hat{Q} \text{ for some } q \in F \big\} \\ \tilde{B}_{acc} \quad &:= \quad \big\{ (\hat{q}, \overline{p}_1, \overline{p}_2, .., \overline{p}_t) \in \tilde{A} \;\big|\; \hat{q} \text{ is of the form } (q, \overline{p}, \overline{y}) \in \hat{Q} \text{ for some } q \in F_{acc} \big\}. \end{aligned}$$

I.e., a state of M is *final* (resp., *accepting*) if, and only if, the associated state of the Turing machine T is. This completes Step 3. ⊣

Step 4: *Definition of M's transition function* $\alpha \;:\; (A \setminus B) \times \big(\mathbb{A}^*\big)^t \times C \;\to\; (A \times \textit{Movement}^t)$.

We let

$$\begin{aligned} \textit{Conf}_T \;:=\; \Big\{ (q, p_1, .., p_{t+u}, w_1, .., w_{t+u}) \;\Big|\; & q \in Q, \text{ and} \\ & \text{for all } j \in \{1,..,t+u\},\; p_j \in \mathbb{N}, \\ & \text{for all } j \in \{1,..,t\},\; w_j \in \{\circledast\}^* \Sigma^* \{\circledast\}^* \text{ with } w_{j,p_j} \in \Sigma, \\ & \text{for all } j \in \{1,..,u\},\; w_{t+j} \in \Sigma^* \Big\}, \end{aligned}$$

where $\circledast$ is a symbol *not* in Σ, and w_{j,p_j} denotes the p_j-th letter in the string w_j.

Intended meaning: The symbol $\circledast$ is used as a *wildcard* symbol that may be interpreted by any symbol in Σ. An element in $\textit{Conf}_T$ gives (potentially) incomplete information on a configuration of T, where the contents of tapes $1,..,t$ might be described only in some part (namely, in the part containing no $\circledast$-symbols).

We let $\tilde{\mathbb{A}} := I \cup C \cup \tilde{A} \cup \{\langle, \rangle\}$. By induction on i we fix, for $i \geq 0$,

- a set $A_i \subseteq \tilde{A}$
- a set $K_i \subseteq (\tilde{A} \setminus \tilde{B}) \times (\tilde{\mathbb{A}}^*)^t$,
- a set $L_i \subseteq \tilde{\mathbb{A}}^*$, letting

$$L_i \quad:=\quad \big\{\; a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle \;:\; (a, y_1, .., y_t) \in K_i \text{ and } c \in C \;\big\} \tag{7}$$

- a function

$$\textit{config}_i : K_i \;\to\; \textit{Conf}_T \cup \{\bot\}$$

  Intended meaning: When the NLM M is in a situation $\kappa \in K_i$, then $\textit{config}_i(\kappa)$ is the Turing machine's configuration at the beginning of M's current step. If $\textit{config}_i(\kappa) = \bot$, then κ does not represent a configuration of the Turing machine.
- the transition function α of M, restricted to K_i, that is,

$$\alpha_{|K_i} : K_i \times C \;\to\; \tilde{A} \times \text{Movement}^t$$

- for every tape $j \in \{1,..,t\}$, a function

$$\textit{tape-config}_{j,i} \;:\; L_i \;\to\; \left\{ (w, p^{[\![}, p^{]\!]}) \;\middle|\; \begin{array}{l} \text{either } 1 \leq p^{[\![} \leq p^{]\!]} \leq \ell \text{ and} \\ \quad w \in \{\circledast\}^{p^{[\![}-1} \Sigma^{p^{]\!]}-p^{[\![}+1} \{\circledast\}^{\ell(N)-p^{]\!]}} \\ \text{or } p^{[\![} > p^{]\!]} \text{ and } w = \varepsilon \end{array} \right\}$$

Intended meaning: When the NLM M is in a situation $\kappa = (a, \langle y_1\rangle, .., \langle y_t\rangle) \in K_i$ and nondeterministically chooses $c \in C$ for its current transition, then

$$\textit{tape-config}_{j,i}\big(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle\big)$$

gives information on the inscription from tape cell $p^{[\![}$ up to tape cell $p^{]\!]}$ of the j-th tape of the Turing machine's configuration at the end of M's current step.

Induction base ($i = 0$): We start with M's start state a_0 and choose

$$A_0 := \{\, a_0 \,\}.$$

If a_0 is *final*, then we let $K_0 := \emptyset$ and $A := A_0$. This then gives us an NLM M which accepts its input without performing a single step. This is fine, since a_0 is final if, and only if, the Turing machine T's start state q_0 is final, that is, T accepts its input without performing a single step.

For the case that a_0 is *not* final, we let

$$K_0 \;:=\; \Big\{\; (a_0, y_1, .., y_t) \;\Big|\; y_1 \in \{\langle v\rangle : v \in I\} \text{ and } y_2 = \cdots = y_t = \langle\rangle \;\Big\}.$$

The set L_0 is defined via equation (7).

The function $\textit{config}_0$ is defined as follows: For every

$$\kappa \;=\; (a_0, y_1, .., y_t) \;\in\; K_0$$

with $y_1 = \langle v\rangle$ (for some $v \in I$), let

$$\textit{config}_0(\kappa) \;:=\; \big(q_0, \overbrace{1, .., 1}^{t+u},\; v\#\circledast^{\ell-(n+1)}, \underbrace{\square^\ell, .., \square^\ell}_{t-1},\; \underbrace{\square^s, .., \square^s}_{u}\big).$$

Let

$$\big(\hat{q}_0, \overline{p}_1, .., \overline{p}_t\big) \;:=\; a_0$$

with $\overline{p}_j = (p_j^{[\![}, p_j^{\uparrow}, p_j^{]\!]}, \textit{head-direction}_j)$, for all $j \in \{1, .., t\}$.

For $j \in \{1, .., t\}$ we define

$$\big(\hat{p}_j^{[\![},\; p_j^{\uparrow},\; \hat{p}_j^{]\!]}\big) := \big(p_j^{[\![}, p_j^{\uparrow}, p_j^{]\!]}\big).$$

Now let $c = (c_1, c_2, .., c_\ell) \in C = C_T^\ell$ be an arbitrary element from M's set C of nondeterministic choices. For defining $\alpha_{|K_0}(\kappa, c)$ and $\textit{tape-config}_{j,0}(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle)$, consider the following: Let us start the Turing machine T with a configuration γ_1 that *fits* to $\textit{config}_0(\kappa)$, i.e., that can be obtained from $\textit{config}_0(\kappa)$ by replacing each occurrence of the wildcard symbol $\circledast$ by an arbitrary symbol in Σ. Let $\gamma_1, \gamma_2, \gamma_3, \ldots$ be the successive configurations of T when started in γ_1 and using the nondeterministic choices $c_1, c_2, c_3, ..$ (in the sense of Definition 17). I.e., for all $\nu \geq 1$, $\gamma_{\nu+1}$ is the $\big(c_\nu \bmod |\mathrm{Next}_T(\gamma_\nu)|\big)$-th of the $|\mathrm{Next}_T(\gamma_\nu)|$ possible next configurations of γ_ν.

Using this notation, the definition of $\alpha_{|K_0}(\kappa, c)$ and

$$\textit{tape-config}_{j,0}(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle)$$

can be taken verbatim from the definition of $\alpha_{|K_{i+1}}(\kappa, c)$ and

$$\textit{tape-config}_{j,i+1}(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle),$$

given below. This completes the induction base ($i = 0$).

Induction step ($i \to i{+}1$): We let

$$A_{i+1} \;:=\; \left\{\; b \in \tilde{A} \;\middle|\; \begin{array}{l} \text{there are } \kappa \in K_i \text{ and } c \in C \text{ such that } \alpha_{|K_i}(\kappa, c) = (b, e_1, .., e_t) \\ \text{(for suitable } (e_1, .., e_t) \in \mathrm{Movement}^t) \end{array} \right\}$$

and

$$K_{i+1} \;:=\; \left\{\; (a, y_1, .., y_t) \;\middle|\; \begin{array}{l} a \in A_{i+1} \setminus \tilde{B}, \\ y_1 \in \{\langle v\rangle : v \in I\} \cup \bigcup_{i' \leq i} L_{i'}, \text{ and} \\ y_j \in \{\langle\rangle\} \cup \bigcup_{i' \leq i} L_{i'}, \text{ for all } j \in \{2, .., t\} \end{array} \right\}$$

The set L_{i+1} is defined via equation (7).

The function $\textit{config}_{i+1}$ is defined as follows: Let $c \in C$ and let $\kappa = (a, y_1, .., y_t) \in K_{i+1}$. Let

$$\big(\hat{q}, \overline{p}_1, .., \overline{p}_t\big) \;:=\; a$$

with $\overline{p}_j = (p_j^{[\![}, p_j^{\uparrow}, p_j^{]\!]}, \mathit{head\text{-}direction}_j)$, for all $j \in \{1,..,t\}$, and

$$\hat{q} \;=\; (q, p_{t+1},.., p_{t+u}, w_{t+1},.., w_{t+u}).$$

Let $j \in \{1,..,t\}$.

If $y_j \in L_{i'}$ for some $i' \leq i$, then let

$$(w'_j, p'^{[\![}_j, p'^{]\!]}_j) \;:=\; \mathit{tape\text{-}config}_{j,i'}(y_j).$$

We choose $w_j := w'_j$. (This is well-defined, because $\mathit{tape\text{-}config}_{j,i'}$ and $\mathit{tape\text{-}config}_{j,i''}$ operate identically on all elements in $L_{i'} \cap L_{i''}$, for all $i', i'' \leq i$).
Furthermore, we let $(\hat{p}_j^{[\![}, \hat{p}_j^{]\!]})$ be defined as follows:

$$(\hat{p}_j^{[\![}, \hat{p}_j^{]\!]}) := \begin{cases} (p_j^{\uparrow}, p'^{]\!]}_j) & \text{if } p_j^{[\![} = p_j^{]\!]} = \ominus \text{ and } \mathit{head\text{-}direction}_j = +1 \\ (p'^{[\![}_j, p_j^{\uparrow}) & \text{if } p_j^{[\![} = p_j^{]\!]} = \ominus \text{ and } \mathit{head\text{-}direction}_j = -1 \\ (p_j^{[\![}, p_j^{]\!]}) & \text{otherwise.} \end{cases}$$

If $y_j \notin \cup_{i' \leq i} L_{i'}$, then we make a case distinction on j: In case that $j \in \{2,..,t\}$, we have $y_j = \langle\rangle$ and $\mathit{head\text{-}direction}_j = +1$. We define $(\hat{p}_j^{[\![}, \hat{p}_j^{]\!]})$ as follows:

$$\left(\hat{p}_j^{[\![}, \hat{p}_j^{]\!]}\right) \;:=\; \left(p_j^{\uparrow}, \ell\right),$$

and choose

$$w_j \;:=\; \circledast^{\hat{p}_j^{[\![}-1} \square^{\ell-(\hat{p}_j^{[\![}-1)}.$$

In case that $j = 1$, we know that y_j must be of the form $\langle v \rangle$, for some $v \in I$, and that $\mathit{head\text{-}direction}_j = +1$. If v is *not* the m-th input item, that is, there is some $\mu \in \{1,..,m-1\}$ such that $(\mu-1)\cdot(n+1) < p_1^{\uparrow} \leq \mu \cdot (n+1)$, then we define

$$\left(\hat{p}_1^{[\![}, \hat{p}_1^{]\!]}\right) \;:=\; \left(p_1^{\uparrow}, \mu \cdot (n+1)\right),$$

and choose

$$w_1 \;:=\; \circledast^{(\mu-1)\cdot(n+1)}\, v\# \circledast^{\ell-\mu\cdot(n+1)}.$$

Otherwise, v must be the m-th input item, that is,

$$p_1^{\uparrow} \;>\; (m-1)\cdot(n+1).$$

In this case we define

$$\left(\hat{p}_1^{[\![}, \hat{p}_1^{]\!]}\right) \;:=\; \left(p_1^{\uparrow}, \ell\right)$$

and choose

$$w_1 \;:=\; \circledast^{(m-1)(n+1)}\, v\# \square^{\ell-m\cdot(n+1)}.$$

If, for some $j_0 \in \{1,..,t\}$, $w_{j_0} = \varepsilon$, then we define

$$\begin{aligned} \mathit{config}_{i+1}(\kappa) \;&:=\; \bot, \\ \mathit{tape\text{-}config}_{j,i+1}\big(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle\big) \;&:=\; (\varepsilon, 2, 1), \end{aligned}$$

and $\alpha_{|K_{i+1}}(\kappa, c) := \big(a, e''_1, .., e''_t\big)$, where for all $j \in \{1,..,t\}$,

$$e''_j \;:=\; \begin{cases} \big(\mathit{head\text{-}direction}_j, \mathit{true}\big) & \text{if } w_j = \varepsilon \\ \big(\mathit{head\text{-}direction}_j, \mathit{false}\big) & \text{otherwise.} \end{cases}$$

In what follows, we consider the case where $w_j \neq \varepsilon$, for all $j \in \{1,..,t\}$. We define

$$\mathit{config}_{i+1}(\kappa) \;:=\; \big(q, p_1,.., p_t, p_{t+1},.., p_{t+u}, w_1,.., w_t, w_{t+1},.., w_{t+u}\big),$$

where q and $p_{t+1},.., p_{t+u}, w_{t+1},.., w_{t+u}$ are obtained from $\hat{q}$,
$p_1,.., p_t$ are obtained from a via $p_j := p_j^{\uparrow}$, for all $j \in \{1,..,t\}$, and
$w_1,.., w_t$ are chosen as above.

Altogether, the description of the definition of $\mathit{config}_{i+1}(\kappa)$ is complete.

For defining $\alpha_{|K_{i+1}}(\kappa, c)$ and $\mathit{tape\text{-}config}_{j,i+1}(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle)$, consider the following:

Let us start the Turing machine T with a configuration γ_1 that *fits* to $\mathit{config}_{i+1}(\kappa)$, i.e., that can be obtained from $\mathit{config}_{i+1}(\kappa)$ by replacing each occurrence of the wildcard symbol $\circledast$ by an arbitrary symbol in Σ. Letting

$$c \;=\; (c_1, c_2, .., c_\ell) \;\in\; C \;=\; C_T^\ell,$$

we let $\gamma_1, \gamma_2, \gamma_3, \ldots$ be the successive configurations of T when started in γ_1 and using the nondeterministic choices $c_1, c_2, c_3, ..$ (in the sense of Definition 17). I.e., for all $\nu \geq 1$, $\gamma_{\nu+1}$ is the $\big(c_\nu \bmod |\mathrm{Next}_T(\gamma_\nu)|\big)$-th of the $|\mathrm{Next}_T(\gamma_\nu)|$ possible next configurations of γ_ν.

Then, there is a minimal $\nu > 1$ for which there exists a $j_0 \in \{1, .., t, \perp\}$ such that throughout the run $\gamma_1 \cdots \gamma_{\nu-1}$,

(1) none of the heads $1, .., t$ changes its direction, and

(2) none of the heads $j \in \{1, .., t\}$ crosses a border $\hat{p}_j^{[\![}$ or $\hat{p}_j^{]\!]}$,

and one of the following cases applies:

Case 1: $j_0 \neq \perp$, and in the transition from $\gamma_{\nu-1}$ to γ_ν, head j_0 crosses one of the borders $\hat{p}_{j_0}^{[\![}$ or $\hat{p}_{j_0}^{]\!]}$. That is, in γ_ν, the j_0-th head is either at position $\hat{p}_{j_0}^{[\![} - 1$ or at position $\hat{p}_{j_0}^{]\!]} + 1$.
(And none of the heads $j \in \{1, .., t\} \setminus \{j_0\}$ crosses a border or changes its direction.[3])

Case 2: $j_0 \neq \perp$, and in the transition from $\gamma_{\nu-1}$ to γ_ν, head j_0 changes its direction, but does not cross one of the borders $\hat{p}_{j_0}^{[\![}$ or $\hat{p}_{j_0}^{]\!]}$.
(And none of the heads $j \in \{1, .., t\} \setminus \{j_0\}$ crosses a border or changes its direction.)

Case 3: γ_ν is *final* and none of the cases 1 and 2 apply. Then we let $j_0 := \perp$.

In all three cases we let

$$(q'', p_1'', .., p_{t+u}'', w_1'', .., w_{t+u}'') \quad := \quad \gamma_\nu \;.$$

We choose

$$\hat{q}'' \quad := \quad (q'', p_{t+1}'', .., p_{t+u}'', w_{t+1}'', .., w_{t+u}'')$$

and define

$$b \quad := \quad (\hat{q}'', \overline{p}_1'', .., \overline{p}_t'') \;,$$

where

$$\overline{p}_j'' \quad = \quad (p''^{[\![}_j, p''^{\uparrow}_j, p''^{]\!]}_j, \mathit{head\text{-}direction}_j'')$$

will be specified below.
Finally, we define

$$\alpha_{|K_{i+1}}(\kappa, c) \quad := \quad (b, e_1'', .., e_t''),$$

where, for every $j \in \{1, .., t\}$,

$$e_j'' \quad := \quad \big(\mathit{head\text{-}direction}_j'',\, \mathit{move}_j''\big)$$

will be specified below.

Recall that $\kappa = \big(a, y_1, .., y_t\big) \in K_{i+1}$. For every $j \in \{1, .., t\}$ we define

$$\mathit{tape\text{-}config}_{j,i+1}\big(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle\big) \;:=\; \begin{cases} \big(\circledast^{p_j^{[\![}-1}\, w''_{j,p_j^{[\![}} \cdots w''_{j,p_j^{]\!]}}\, \circledast^{\ell - p_j^{]\!]}+1},\; p_j^{[\![},\; p_j^{]\!]}\big) & \text{if } p_j^{[\![} \leq p_j^{]\!]} \\ \big(\varepsilon, p_j^{[\![}, p_j^{]\!]}\big) & \text{otherwise} \end{cases}$$

where $p_j^{[\![}$ and $p_j^{]\!]}$ are specified below.

For all $j \in \{1, .., t\} \setminus \{j_0\}$ we know (by the choice of ν and j_0) that throughout the Turing machine's computation $\gamma_0, .., \gamma_\nu$, head j neither changes its direction nor crosses one of the borders $\hat{p}_j^{[\![}, \hat{p}_j^{]\!]}$. Consequently, we

[3] Recall that w.l.o.g. we assume that the Turing machine is normalized, cf. Definition 23.

choose

$$\begin{aligned}
\textit{head-direction}''_j &:= \textit{head-direction}_j \\
\textit{move}''_j &:= \textit{false} \\
p''^{\uparrow}_j &:= p''_j \\
p''^{[\![}_j &:= \begin{cases} p''^{\uparrow}_j & \text{if } \textit{head-direction}_j = +1 \\ \hat{p}^{[\![}_j & \text{if } \textit{head-direction}_j = -1 \end{cases} \\
p''^{]\!]}_j &:= \begin{cases} \hat{p}^{]\!]}_j & \text{if } \textit{head-direction}_j = +1 \\ p''^{\uparrow}_j & \text{if } \textit{head-direction}_j = -1 \end{cases} \\
p^{[\![}_j &:= \begin{cases} \hat{p}^{[\![}_j & \text{if } \textit{head-direction}_j = +1 \\ p''^{\uparrow}_j + 1 & \text{if } \textit{head-direction}_j = -1 \end{cases} \\
p^{]\!]}_j &:= \begin{cases} p''^{\uparrow}_j - 1 & \text{if } \textit{head-direction}_j = +1 \\ p''^{]\!]}_j & \text{if } \textit{head-direction}_j = -1 \end{cases}
\end{aligned}$$

In **Case 3** we have $j_0 = \perp$, and therefore, $\alpha_{|K_{i+1}}(\kappa, c)$ and $\textit{tape-config}_{j,i+1}(a\langle y_1\rangle \cdots \langle y_t\rangle\langle c\rangle)$ is fully specified. Furthermore, note that in Case 3 we know that γ_ν is *final*, i.e., q'' is a final state of the Turing machine T. Therefore, b is a *final* state of the NLM M, and M's run accepts if, and only if, the simulated Turing machine run accepts (recall the definition of M's set of final and accepting states at the end of Step 3).

For **Case 1** and **Case 2**, we have $j_0 \in \{1,\ldots,t\}$, and for specifying

$$\textit{head-direction}''_{j_0},\ \textit{move}''_{j_0},\ p''^{[\![}_{j_0},\ p''^{\uparrow}_{j_0},\ p''^{]\!]}_{j_0},\ p^{[\![}_{j_0},\ \textit{and}\ p^{]\!]}_{j_0},$$

we distinguish between the two cases:

ad Case 1: In this case, $j_0 \neq \perp$, and head j_0 crosses one of the borders $\hat{p}^{[\![}_{j_0}$ or $\hat{p}^{]\!]}_{j_0}$ in the transition from $\gamma_{\nu-1}$ to γ_ν (that is, p''_{j_0} is either $\hat{p}^{]\!]}_{j_0} + 1$ or $\hat{p}^{[\![}_{j_0} - 1$). We choose

$$\begin{aligned}
\left(p''^{[\![}_{j_0},\ p''^{\uparrow}_{j_0},\ p''^{]\!]}_{j_0}\right) &:= \left(\ominus,\ p''_{j_0},\ \ominus\right) \\
\left(p^{[\![}_{j_0},\ p^{]\!]}_{j_0}\right) &:= \left(\hat{p}^{[\![}_{j_0},\ \hat{p}^{]\!]}_{j_0}\right) \\
\textit{move}''_{j_0} &:= \textit{true} \\
\textit{head-direction}''_{j_0} &:= \begin{cases} +1 & \text{if } p''_{j_0} = \hat{p}^{]\!]}_{j_0} + 1 \\ -1 & \text{otherwise.} \end{cases}
\end{aligned}$$

ad Case 2: In this case, $j_0 \neq \perp$, and head j_0 changes its direction, but does not cross one of the borders $\hat{p}^{[\![}_{j_0}$ or $\hat{p}^{]\!]}_{j_0}$. We only consider the case where the direction of head j_0 changes from $+1$ to -1 (the other case is symmetric).
We choose

$$\begin{aligned}
\left(\textit{head-direction}''_{j_0},\ \textit{move}''_{j_0}\right) &:= \left(-1, \textit{false}\right) \\
\left(p''^{[\![}_{j_0},\ p''^{\uparrow}_{j_0},\ p''^{]\!]}_{j_0}\right) &:= \left(\hat{p}^{[\![}_{j_0},\ p''_{j_0},\ p''_{j_0} + 1\right) \\
\left(p^{[\![}_{j_0},\ p^{]\!]}_{j_0}\right) &:= \left(p''_{j_0} + 2,\ \hat{p}^{]\!]}_{j_0}\right)
\end{aligned}$$

Note that here we might have $p''_{j_0} + 1 = \hat{p}^{]\!]}_{j_0}$. In this case, by the above definition, we obtain $p^{[\![}_{j_0} = p^{]\!]}_{j_0} + 1$.

Altogether, this completes the induction step.

Finally, we are ready to fix M's state set A and transition function α as follows:

$$\begin{aligned}
A &:= \bigcup_{i\geq 0} A_i \\
K &:= \bigcup_{i\geq 0} K_i \\
\alpha &:= \bigcup_{i\geq 0} \alpha_{|K_i}
\end{aligned}$$

Note that

1. α is well-defined, because $\alpha_{|K_i}$ and $\alpha_{|K_{i'}}$ operate identical on all elements in $(K_i \cap K_{i'}) \times C$ (for all $i, i' \geq 0$).
2. K consists of all situations $(a, y_1, .., y_t) \in (A \setminus B) \times (\mathbb{A}^*)^t$ that may occur in runs of M.
3. α remains undefined for elements $(a, y_1, .., y_t)$ in $(A \setminus B) \times (\mathbb{A}^*)^t$ that do *not* belong to K. This is fine, because such a situation $(a, y_1, .., y_t)$ can never occur in an actual run of M.

This completes Step 4. ⊣

Note that finally, the NLM M is fully specified. Due to the construction we know that M is (r,t)-bounded, because it has t lists and the number of head reversals during each run on an input $\overline{v} = (v_1, .., v_m) \in I^m$ is bounded by the number $r-1 = r(m \cdot (n+1))-1$ of head reversals of the according run of the Turing machine T on input $v_1 \# \cdots v_m \#$.

Step 5: *For every input $\overline{v} = (v_1, .., v_m) \in I^m$ we have*

$$\Pr(M \text{ accepts } \overline{v}) \;=\; \Pr\big(T \text{ accepts } v_1 \# \cdots \# v_m\big).$$

Proof: Let $\ell_M \in \mathbb{N}$ be an upper bound on the length of runs of the NLM M (such a number ℓ_M exists, because M is (r,t)-bounded; see Lemma 31 (a) in Appendix D).

For the remainder of this proof we fix an input $\overline{v} = (v_1, .., v_m) \in I^m$ for the NLM M and we let $\tilde{v} := v_1 \# \cdots v_m \#$ denote the corresponding input for the Turing machine T.

From Lemma 18 we know that

$$\Pr(T \text{ accepts } \tilde{v}) \;=\; \frac{|\{\overline{c}_T \in C_T^\ell : \rho_T(\tilde{v}, \overline{c}_T) \text{ accepts}\}|}{|C_T^\ell|} \;=\; \frac{|\{\overline{c}_T \in C_T^\ell : \rho_T(\tilde{v}, \overline{c}_T) \text{ accepts}\}|}{|C|}.$$

Furthermore, we know from Lemma 25 that

$$\Pr(M \text{ accepts } \overline{v}) \;=\; \frac{|\{\overline{c} \in C^{\ell_M} : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}|}{|C|^{\ell_M}}.$$

For showing that $\Pr(M \text{ accepts } \overline{v}) = \Pr\big(T \text{ accepts } \tilde{v}\big)$ it therefore suffices to show that

$$|\{\overline{c} \in C^{\ell_M} : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}| \;=\; |C|^{\ell_M - 1} \cdot |\{\overline{c}_T \in C_T^\ell : \rho_T(\tilde{v}, \overline{c}_T) \text{ accepts}\}|.$$

Consequently, it suffices to show that there is a function

$$f \;:\; C^{\ell_M} \;\to\; C_T^\ell$$

such that

- for every $\overline{c} \in C^{\ell_M}$, the list machine run $\rho_M(\overline{v}, \overline{c})$ simulates the Turing machine run $\rho_T(\tilde{v}, f(\overline{c}))$, and
- for every $\overline{c}_T \in C_T^\ell$,

$$|\{\overline{c} \in C^{\ell_M} : f(\overline{c}) = \overline{c}_T\}| \;=\; |C|^{\ell_M - 1}. \tag{8}$$

We can define such a function f as follows:

For every sequence

$$\overline{c} \;=\; \big(c^{(1)}, \ldots, c^{(\ell_M)}\big) \;\in\; C^{\ell_M},$$

following the construction of the NLM M in Steps 1–4, we obtain for each $i \in \{1, .., \ell_M\}$ that there is a uniquely defined prefix $\tilde{c}^{(i)}$ of M's nondeterministic choice

$$c^{(i)} \;=\; \big(c_1^{(i)}, .., c_\ell^{(i)}\big) \;\in\; C \;=\; C_T^\ell,$$

such that the following is true for

$$\tilde{c} \;:=\; \tilde{c}^{(1)} \tilde{c}^{(2)} \cdots \tilde{c}^{(\ell_M)},$$

viewed as a sequence of elements from C_T:

(1) The list machine run $\rho_M(\overline{v}, \overline{c})$ simulates the Turing machine run $\rho_T(\tilde{v}, \tilde{c})$, where M uses in its i-th step exactly the $\tilde{c}^{(i)}$-portion of $c^{(i)}$ for simulating the according Turing machine steps.

(2) If $\tilde{\ell} \leq \ell$ denotes the length of the run $\rho_T(\tilde{v}, \tilde{c}) = (\rho_1, .., \rho_{\tilde{\ell}})$, then $\tilde{c}$ has exactly the length $\tilde{\ell}-1$.

Now let i_0 denote the maximum element from $\{1, .., \ell_M\}$ such that $|\tilde{c}^{(i_0)}| \neq 0$ (in particular, this implies that $\tilde{c} = \tilde{c}^{(1)} \cdots \tilde{c}^{(i_0)}$). We let $\tilde{\tilde{c}}^{(i_0)}$ be the prefix of $c^{(i_0)}$ of length $\ell - (\tilde{\ell} - 1 - |\tilde{c}^{(i_0)}|)$ and define

$$\tilde{\tilde{c}} \;:=\; \tilde{c}^{(1)} \cdots \tilde{c}^{(i_0 - 1)} \tilde{\tilde{c}}^{i_0}.$$

Note that, viewed as a sequence of elements from C_T, $\tilde{\tilde{c}}$ has length exactly ℓ, and therefore, we can well define

$$f(\overline{c}) \;:=\; \tilde{\tilde{c}}.$$

Furthermore, to see that (8) is satisfied, note that f is surjective, i.e., for every $\tilde{\tilde{c}} \in C_T^{\ell}$ there exists a $\overline{c}$ with $f(\overline{c}) = \tilde{\tilde{c}}$, and

$$|\{\overline{c} \in C^{\ell_M} : f(\overline{c}) = \tilde{\tilde{c}}\}| \;=\; |C_T|^{\ell \cdot \ell_M - \ell} \;=\; |C_T|^{\ell \cdot (\ell_M - 1)} \;=\; |C|^{\ell_M - 1}.$$

(For the first equation, note that through $\tilde{\tilde{c}}$, exactly ℓ of the possible $\ell \cdot \ell_M$ C_T-components of $\overline{c}$ are fixed, whereas each of the remaining $\ell \cdot \ell_M - \ell$ components may carry an arbitrary element from C_T.)
This completes Step 5. ⊣

Altogether, the proof of Lemma 16 is complete. □

D. DETAILED PROOF OF LEMMA 21

This section is devoted to the proof of Lemma 21.

After pointing out an easy observation concerning randomized list machines in subsection D.1, we formally fix the notion of the *skeleton* of a list machine's run in subsection D.2. Then, in subsection D.3 we state and prove some basic properties of list machines concerning the size and shape of runs and the possibility of composing different runs. Afterwards, in subsection D.4, we take a closer look at the information flow that can occur during a list machine's computation, and we show that only a small number of input positions can be compared during an NLM's run. Finally, in subsection D.5, we prove Lemma 21.

D.1 An Easy Observation Concerning Randomized List Machines

Lemma 26. *Let $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ be an NLM, let ℓ be an upper bound on the length of M's runs, and let $\mathscr{J} \subseteq I^m$ such that* $\Pr(M \text{ accepts } \overline{v}) \geq \frac{1}{2}$, *for all inputs $\overline{v} \in \mathscr{J}$. Then there is a sequence $\overline{c} = (c_1, \ldots, c_\ell) \in C^\ell$ such that the set*

$$\mathscr{J}_{acc,\overline{c}} \;:=\; \{\overline{v} \in \mathscr{J} : \rho_M(\overline{v}, \overline{c}) \textit{ accepts}\}$$

has size $|\mathscr{J}_{acc,\overline{c}}| \geq \frac{1}{2} \cdot |\mathscr{J}|$.

Proof: By assumption we know that

$$\sum_{\overline{v} \in \mathscr{J}} \Pr(M \text{ accepts } \overline{v}) \;\geq\; |\mathscr{J}| \cdot \frac{1}{2}.$$

From Lemma 25 we obtain

$$\sum_{\overline{v} \in \mathscr{J}} \Pr(M \text{ accepts } \overline{v}) \;=\; \sum_{\overline{v} \in \mathscr{J}} \frac{|\{\overline{c} \in C^\ell : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}|}{|C^\ell|}.$$

Therefore,

$$\sum_{\overline{v} \in \mathscr{J}} |\{\overline{c} \in C^\ell : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}| \;\geq\; |C^\ell| \cdot \frac{|\mathscr{J}|}{2}.$$

On the other hand,

$$\sum_{\overline{v} \in \mathscr{J}} |\{\overline{c} \in C^\ell : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}| \;=\; \sum_{\overline{c} \in C^\ell} |\{\overline{v} \in \mathscr{J} : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}|.$$

Consequently,

$$\sum_{\overline{c} \in C^\ell} |\{\overline{v} \in \mathscr{J} : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}| \;\geq\; |C^\ell| \cdot \frac{|\mathscr{J}|}{2}.$$

Therefore, there must exist at least one $\overline{c} \in C^\ell$ with

$$|\{\overline{v} \in \mathscr{J} : \rho_M(\overline{v}, \overline{c}) \text{ accepts}\}| \;\geq\; \frac{|\mathscr{J}|}{2},$$

and the proof of Lemma 26 is complete. □

D.2 Skeletons of runs

Definition 27 (local_views(ρ), ndet_choices(ρ), moves(ρ)). Let M be an NLM.

(a) The *local view*, $lv(\gamma)$, of a configuration $\gamma = (a, p, d, X)$ of M is defined via

$$lv(\gamma) \;:=\; (a, d, y) \quad \text{with} \quad y \;:=\; \begin{pmatrix} x_{1,p_1} \\ \vdots \\ x_{t,p_t} \end{pmatrix}.$$

I.e., $lv(\gamma)$ carries the information on M's current state, head directions, and contents of the list cells currently being seen.

(b) Let $\rho = (\rho_1, .., \rho_\ell)$ be a run of M. We define

(i)
$$\text{local\_views}(\rho) \;:=\; \big(lv(\rho_1), \ldots, lv(\rho_\ell)\big).$$

(ii) $\text{ndet\_choices}(\rho) \subseteq C^{\ell-1}$ to be the set of all sequences $\overline{c} = (c_1, .., c_{\ell-1})$ such that, for all $i < \ell$, ρ_{i+1} is the c_i-successor of ρ_i.

Note that $\Pr(\rho) = \frac{|\text{ndet\_choices}(\rho)|}{|C|^{\ell-1}}$.

(iii)
$$\text{moves}(\rho) \;:=\; \big(\text{move}_1, .., \text{move}_{\ell-1}\big) \;\in\; \big(\{0, 1, -1\}^t\big)^{\ell-1},$$

where, for every $i < \ell$, $\text{move}_i = (\text{move}_{i,1}, .., \text{move}_{i,t})^\top \in \{0, 1, -1\}^t$ such that, for each $\tau \in \{1, .., t\}$, $\text{move}_{i,\tau} = 0$ (resp., 1, resp., -1) if, and only if, in the transition from configuration ρ_i to configuration ρ_{i+1}, the head on the τ-th list stayed on the same list cell (resp., moved to the next cell to the right, resp., to the left). ⊣

To prove lower bound results for list machines, we use the notion of a *skeleton* of a run. Basically, a skeleton describes the information *flow* during a run, in the sense that it does *not* describe the exchanged data items (i.e., input values), but instead, it describes which input *positions* the data items originally came from. The input positions of an NLM $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ are simply the indices $i \in \{1, \ldots, m\}$.

Definition 28 (Index Strings and Skeletons). Let M be an NLM, let $\overline{v} = (v_1, .., v_m) \in I^m$ be an input for M, let ρ be a run of M for input $\overline{v}$, and let $\gamma = (a, p, d, X)$ be one of the configurations in ρ.

(a) For every cell content $x_{\tau,j}$ in X (for each list $\tau \in \{1, .., t\}$), we write

$$ind(x_{\tau,j})$$

to denote the *index string*, i.e., the string obtained from $x_{\tau,j}$ by replacing each occurrence of input number v_i by its index (i.e., input position) $i \in \{1, .., m\}$, and by replacing each occurrence of a nondeterministic choice $c \in C$ by the wildcard symbol "?".

(b) For $y = \big(x_{1,p_1}, .., x_{t,p_t}\big)^\top$ we let

$$ind(y) \;:=\; \big(ind(x_{1,p_1}), .., ind(x_{t,p_t})\big)^\top.$$

(c) The *skeleton* of a configuration γ's local view $lv(\gamma) = (a, d, y)$ is defined via

$$skel(lv(\gamma)) \;:=\; \big(a, d, ind(y)\big).$$

(d) The *skeleton of a run* $\rho = (\rho_1, .., \rho_\ell)$ of M is defined via

$$skel(\rho) \;:=\; \big(s, \text{moves}(\rho)\big),$$

where $s = (s_1, .., s_\ell)$ with $s_1 := skel(lv(\rho_1))$, and for all $i < \ell$, if $\text{moves}(\rho) = (\text{move}_1, .., \text{move}_{\ell-1})^\top$,

$$s_{i+1} \;:=\; \begin{cases} skel(lv(\rho_{i+1})) & \text{if } \text{move}_i \neq (0, 0, .., 0)^\top \\ \text{"?"} & \text{otherwise.} \end{cases}$$

Remark 29. Note that, given an input instance $\overline{v}$ for an NLM M, the skeleton $\zeta := skel(\rho)$ of a run ρ of M on input $\overline{v}$, and a sequence $\overline{c} \in \text{ndet\_choices}(\rho)$, the entire run ρ can be reconstructed. ⊣

D.3 Basic Properties of List Machines

In this section we provide some basic properties of list machines concerning the size and shape of runs, the number of skeletons of runs, and the possibility of composing different runs.

Lemma 30 (List length and cell size).
Let $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ be an (r,t)-bounded NLM.

(a) The *total list length* of a configuration of M is defined as the sum of the lengths (i.e., number of cells) of all lists in that configuration.[4]
For every $i \in \{1,..,r\}$, the total list length of each configuration that occurs before the i-th change of a head direction is $\leq (t+1)^i \cdot m$.
In particular, the total list length of each configuration in each run of M is $\leq (t+1)^r \cdot m$.

(b) The *cell size* of a configuration of M is defined as the maximum length of the entries of the cells occurring in the configuration (remember that the cell entries are strings over $\mathbb{A} = I \cup C \cup A \cup \{\langle, \rangle\}$).
The cell size of each configuration in each run of M is $\leq 11 \cdot (\max\{t,2\})^r$.

Proof: For *deterministic* list machines, (a) and (b) were proved in [10] (cf., Claims 1 and 2 in the proof of [10, Lemma 15]). For *nondeterministic* list machines, the proofs are virtually identical; only the cell size increases, as now the list entries also contain the nondeterministic choices.

In fact, the proof of (a) is identical to the proof of [10, Claim 2 in the proof of Lemma 15]: Let γ be a configuration of total list length ℓ. Then the total list length of a successor configuration γ' of γ is at most $\ell + t$, if a head moves or changes its direction in the transition from γ to γ', and it remains ℓ otherwise.
Now suppose γ' is a configuration that can be reached from γ without changing the direction of any head. Then γ' is reached from γ with at most $\ell - t$ head movements, because a head can move into the same direction for at most $\lambda - 1$ times on a list of length λ. Thus the total list length of γ' is at most

$$\ell + t \cdot (\ell - t). \tag{9}$$

The total list length of the initial configuration is $m + t - 1$. A simple induction based on (9) shows that the total list length of a configuration that occurs before the i-th change of a head direction is at most

$$(t+1)^i \cdot m.$$

This proves (a).

For the proof of (b), let γ be a configuration of cell size s. Then the cell size of all configurations that can be reached from γ without changing the direction of any head is at most

$$1 + t \cdot (2+s) + 3 \;=\; 4 + t \cdot (2+s).$$

The cell size of the initial configuration is 3. A simple induction shows that the total cell size of any configuration that occurs before the i-th change of a head direction is at most

$$4 + \sum_{j=1}^{i-1} 6t^j + 5t^i \;\leq\; 11 \cdot (\max\{t,2\})^i.$$

□

Lemma 31 (The shape of runs of an NLM). *Let $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ be an (r,t)-bounded NLM, and let $k := |A|$. The following is true for every run $\rho = (\rho_1, .., \rho_\ell)$ of M and the corresponding sequence $moves(\rho) = (move_1, .., move_{\ell-1})$.*

(a) $\ell \leq k + k \cdot (t+1)^{r+1} \cdot m$.

(b) There is a number $\mu \leq (t+1)^{r+1} \cdot m$ and there are indices $1 \leq j_1 < j_2 < \cdots < j_\mu < \ell$ such that:

(i) For every $i \in \{1,..,\ell-1\}$,

$$move_i \neq (0,0,..,0)^\top \iff i \in \{j_1, .., j_\mu\}.$$

(ii) If $\mu = 0$, then $\ell \leq k$.
Otherwise, $j_1 \leq k$; $j_{\nu+1} - j_\nu \leq k$, for every $\nu \in \{1,..,\mu-1\}$; and $\ell - j_\mu \leq k$.

Proof: For indices $i < \ell$ with $\text{move}_i = (0,0,..,0)^\top$ we know from Definition 24 (c) that the *state* is the only thing in which ρ_i and ρ_{i+1} my differ. As (r,t)-bounded NLMs are *not* allowed to have an *infinite* run, we obtain that without moving any of its heads, M can make at most k consecutive steps.

On the other hand, for every $i \in \{1,..,r\}$ we know from Lemma 30 (a) that the total list length of a configuration that occurs before the i-th change of a head direction is

$$\leq\; (t+1)^i \cdot m. \tag{10}$$

Thus, between the $(i-1)$-st and the i-th change of a head direction, the number of steps in which at least one head moves is

$$\leq (t+1)^i \cdot m.$$

[4] Note that the total list length never decreases during a computation.

Altogether, for every run ρ of M, the total number of steps in which at least one head moves is

$$\leq \sum_{i=1}^{r}(t+1)^i \cdot m \;\leq\; (t+1)^{r+1}\cdot m. \tag{11}$$

Hence, we obtain that the total length of each run of M is

$$\leq\; k + k\cdot(t+1)^{r+1}\cdot m$$

(namely, M can pass through at most k configurations before moving a head for the first time, it can move a head for at most $(t+1)^{r+1}\cdot m$ times, and between any two head movements, it can pass through at most k configurations).
Altogether, the proof of Lemma 31 is complete. □

Lemma 32 (Number of Skeletons).
Let $M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ be an (r,t)-bounded NLM with $t \geq 2$ and $k := |A| \geq 2$.
The number

$$|\{skel(\rho) : \rho \text{ is a run of } M\}|$$

of skeletons of runs of M is

$$\leq \quad \big(m+k+3\big)^{12\cdot m\cdot(t+1)^{2r+2}+24\cdot(t+1)^r}.$$

Proof: We first count the number of skeletons of local views of *configurations* γ of M.
Let γ be a configuration of M, and let $lv(\gamma)$ be of the form (a,d,y). Then,

$$skel(lv(\gamma)) \;=\; (a,d,ind(y)),$$

where $a \in A$, $d \in \{-1,1\}^t$, and $ind(y)$ is a string over the alphabet

$$\{1,..,m\} \cup \{\text{“?”}\} \cup A \cup \{\langle,\rangle\}.$$

Due to Lemma 30 (b), the string $ind(y)$ has length $\leq 11\cdot t^r$. Therefore,

$$|\{skel(lv(\gamma)) : \gamma \text{ is a configuration of } M\}| \;\leq\; k\cdot 2^t\cdot\big(m+k+3\big)^{11\cdot t^r}. \tag{12}$$

From Lemma 31 we know that for every run $\rho = (\rho_1,..,\rho_\ell)$ of M there is a number $\mu \leq (t+1)^{r+1}\cdot m$ and indices $1 \leq j_1 < j_2 < \cdots < j_\mu < \ell$ such that for moves$(\rho) = (\text{move}_1,..,\text{move}_{\ell-1})$ we have:

(i) For every $i \in \{1,..,\ell-1\}$, $\text{move}_i \neq (0,0,..,0)^\top \iff i \in \{j_1,..,j_\mu\}$.

(ii) If $\mu = 0$, then $\ell \leq k$.
Otherwise, $j_1 \leq k$; $j_{\nu+1} - j_\nu \leq k$, for every $\nu \in \{1,..,\mu-1\}$; and $\ell - j_\mu \leq k$.

The total number of possibilities of choosing such μ, ℓ, $j_1,..,j_\mu$ is

$$\leq \sum_{\mu=0}^{(t+1)^{r+1}\cdot m} k^{\mu+1} \;\leq\; k^{2+(t+1)^{r+1}\cdot m}. \tag{13}$$

For each fixed ρ with parameters $\mu,\ell,j_1,..,j_\mu$, $skel(\rho) = (s,\text{moves}(\rho))$ is of the following form: For every $i \leq \ell$ with $i \notin \{j_1,..,j_\mu\}$, $\text{move}_i = (0,0,..,0)^\top$ and $s_{i+1} =$ “?”. For the remaining indices $j_1,..,j_\mu$, there are

$$\leq\; 3^{t\cdot\mu} \;\leq\; 3^{(t+1)^{r+2}\cdot m} \tag{14}$$

possibilities of choosing $(\text{move}_{j_1},..,\text{move}_{j_\mu}) \in \big(\{0,1,-1\}^t\big)^\mu$, and there are

$$\leq\; |\{skel(lv(\gamma)) : \gamma \text{ is a configuration of } M\}|^\mu \;\leq\; \Big(k\cdot 2^t\cdot\big(m+k+3\big)^{11\cdot t^r}\Big)^{(t+1)^{r+1}\cdot m} \tag{15}$$

possibilities of choosing $(s_{j_1+1},..,s_{j_\mu+1}) = \big(skel(lv(\rho_{j_1+1})),..,skel(lv(\rho_{j_\mu+1}))\big)$.

In total, by computing the product of the terms in (13), (14), and (15), we obtain that the number $|\{skel(\rho) : \rho \text{ is a run of } M\}|$ of skeletons of runs of M is at most

$$\begin{aligned}
&\Big(k^{2+(t+1)^{r+1}\cdot m}\Big)\cdot\Big(3^{(t+1)^{r+2}\cdot m}\Big)\cdot\Big(\big(k\cdot 2^t\cdot\big(m+k+3\big)^{11\cdot t^r}\big)^{(t+1)^{r+1}\cdot m}\Big)\\
\leq\quad &\Big(k\cdot 3\cdot k\cdot 2^t\cdot\big(m+k+3\big)^{11\cdot t^r}\Big)^{2+(t+1)^{r+2}\cdot m}\\
\leq\quad &\Big(k^2\cdot 2^{t+\log 3}\cdot(m+k+3)^{11\cdot t^r}\Big)^{2+(t+1)^{r+2}\cdot m}.
\end{aligned} \tag{16}$$

Obviously,

$$k^2 \;\leq\; (k+m+3)^2.$$

Since $(k+m+3) \geq 2^2$, we have

$$2^{t+\log 3} \;\leq\; (m+k+3)^{t+1}.$$

Inserting this into (16), we obtain that the number of skeletons of runs of M is

$$\begin{array}{ll} \leq & \big(m+k+3\big)^{(11t^r+t+3)\cdot(2+(t+1)^{r+2}\cdot m)} \\ \leq & \big(m+k+3\big)^{(12\cdot(t+1)^r)\cdot(2+(t+1)^{r+2}\cdot m)} \\ \leq & \big(m+k+3\big)^{24\cdot(t+1)^r+12\cdot(t+1)^{2r+2}\cdot m}. \end{array}$$

This completes the proof of Lemma 32. □

Definition 33. Let $M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ be an NLM and let

$$\zeta \;=\; \big((s_1,..,s_\ell),(\text{move}_1,..,\text{move}_{\ell-1})\big)$$

be the skeleton of a run ρ of M. We say that two input positions $i,i' \in \{1,..,m\}$, are *compared* in ζ (respectively, in ρ) iff there is a $j \leq \ell$ such that s_j is of the form

$$\mathit{skel}(\mathit{lv}(\gamma)) \;=\; (a,d,\mathit{ind}(y)), \quad \text{for some configuration } \gamma,$$

and both i and i' occur in $\mathit{ind}(y)$. ⊣

Lemma 34 (Composition Lemma). *Let $M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ be an NLM and let $\ell \in \mathbb{N}$ be an upper bound on the length of M's runs. Let ζ be the skeleton of a run of M, and let i,i' be input positions of M that are not compared in ζ. Let $\overline{v} = (v_1,..,v_m)$ and $\overline{w} = (w_1,..,w_m)$ be two different inputs for M with*

$$w_j = v_j, \;\; \textit{for all } j \in \{1,..,m\} \setminus \{i,i'\}$$

(i.e., $\overline{v}$ and $\overline{w}$ only differ at the input positions i and i'). Furthermore, suppose there exists a sequence $\overline{c} = (c_1,..,c_\ell) \in C^\ell$ such that

$$\mathit{skel}\big(\rho_M(\overline{v},\overline{c})\big) \;=\; \mathit{skel}\big(\rho_M(\overline{w},\overline{c})\big) \;=\; \zeta,$$

and $\rho_M(\overline{v},\overline{c})$ and $\rho_M(\overline{w},\overline{c})$ either both accept or both reject. Then, for the inputs $\overline{u} := (v_1,..,v_i,..,w_{i'},..,v_m)$ and $\overline{u}' := (v_1,..,w_i,..,v_{i'},..,v_m)$ we have

$$\zeta \;=\; \mathit{skel}\big(\rho_M(\overline{u},\overline{c})\big) \;=\; \mathit{skel}\big(\rho_M(\overline{u}',\overline{c})\big)$$

and

$$\rho_M(\overline{u},\overline{c}) \textit{ accepts} \iff \rho_M(\overline{u}',\overline{c}) \textit{ accepts} \iff \rho_M(\overline{v},\overline{c}) \textit{ accepts} \iff \rho_M(\overline{w},\overline{c}) \textit{ accepts.}$$

Proof: Let $\zeta = ((s_1,\ldots,s_{\ell'}),(\mathit{move}_1,\ldots,\mathit{move}_{\ell'-1}))$ be the skeleton as in the hypothesis of the lemma. We show that $\mathit{skel}(\rho_M(\overline{u},\overline{c})) = \zeta$, and that $\rho_M(\overline{u},\overline{c})$ accepts if and only if $\rho_M(\overline{v},\overline{c})$ and $\rho_M(\overline{w},\overline{c})$ accept. The proof for $\overline{u}'$ instead of $\overline{u}$ is the same.

Let $\mathit{skel}(\rho_M(\overline{u},\overline{c})) = ((s'_1,\ldots,s'_{\ell''}),(\mathit{move}'_1,\ldots,\mathit{move}'_{\ell''-1}))$. Let j be the maximum index such that

(i) $(s'_1,\ldots,s'_j) = (s_1,\ldots,s_j)$, and

(ii) $(\mathit{move}'_1,\ldots,\mathit{move}'_{j-1}) = (\mathit{move}_1,\ldots,\mathit{move}_{j-1})$.

Let j' be the maximum index such that $j' \leq j$ and $s_{j'} = s'_{j'} \neq$ "?". By the hypothesis of the lemma we know that i and i' do not occur both in $s_{j'}$. Thus for some $\overline{x} \in \{\overline{v},\overline{w}\}$, $s_{j'}$ contains only input positions where $\overline{u}$ and $\overline{x}$ coincide. Let $\rho_M(\overline{x},\overline{c}) = (\rho_1,\ldots,\rho_{\ell'})$, and let $\rho_M(\overline{u},\overline{c}) = (\rho'_1,\ldots,\rho'_{\ell''})$. Since $s_{j'}$ contains only input positions where $\overline{u}$ and $\overline{x}$ coincide, we have $\mathit{lv}(\rho_{j'}) = \mathit{lv}(\rho'_{j'})$. Since $\mathit{move}_{j''} = \mathit{move}'_{j''} = (0,\ldots,0)^\top$ for all $j'' \in \{j',\ldots,j-1\}$, we therefore have $\mathit{lv}(\rho_j) = \mathit{lv}(\rho'_j)$. This implies that the behavior in the j-th step of both runs, $\rho_M(\overline{x},\overline{c})$ and $\rho_M(\overline{u},\overline{c})$, is the same.

Case 1 ($j = \ell'$): In this case there is no further step in the run, from which we conclude that $\ell' = \ell''$. Hence both skeletons, ζ and $\mathit{skel}(\rho_M(\overline{u},\overline{c}))$, are equal. Moreover, $\mathit{lv}(\rho_j) = \mathit{lv}(\rho'_j)$ implies that both runs either accept or reject.

Case 2 ($j < \ell'$): In this case we know that $\ell'' \geq j+1$, and that $\mathit{move}_j = \mathit{move}'_j$. By the choice of j we also have $s_{j+1} \neq s'_{j+1}$, which together with $\mathit{move}_j = \mathit{move}'_j$ implies $s_{j+1} \neq$ "?" and $s'_{j+1} \neq$ "?". Let $s_{j+1} = (a,d,\mathit{ind})$ and $s'_{j+1} = (a',d',\mathit{ind}')$. Since $\mathit{lv}(\rho_j) = \mathit{lv}(\rho'_j)$, and the behavior in the j-th step of both runs is the same, we have $a = a'$ and $d = d'$. So, ind and ind' must differ on some component $\tau \in \{1,\ldots,t\}$. Let ind_τ be the τ-th component of ind, and let ind'_τ be the τ-th component of ind'.

Since $(s'_1,\ldots,s'_j) = (s_1,\ldots,s_j)$ and $(\mathit{move}'_1,\ldots,\mathit{move}'_j) = (\mathit{move}_1,\ldots,\mathit{move}_j)$, the list cells visited directly after step $j'' \in \{0,\ldots,j\}$ of all three runs, $\rho_M(\overline{v},\overline{c})$, $\rho_M(\overline{w},\overline{c})$ and $\rho_M(\overline{u},\overline{c})$, are the same. This in particular implies that ind_τ and ind'_τ describe the same list cells, though in different runs. So, if $\mathit{ind}_\tau = \langle p \rangle$ for some input position p, or $\mathit{ind}_\tau = \langle\rangle$, then the cell described by ind_τ has not been visited during the first j steps of

all three runs, and therefore, $ind'_\tau = ind_\tau$. Now we may assume that $ind_\tau \neq \langle p \rangle$ for all input positions p, and $ind_\tau \neq \langle\rangle$. Then, $ind_\tau = a\langle y_1\rangle \dots \langle y_t\rangle\langle c\rangle$, where $(a,d,y_1,\dots,y_t) = s_{j''}$ for some $j'' \in \{1,\dots,j\}$, and c is the j''-th nondeterministic choice of $\overline{c}$. Also, $ind'_\tau = a'\langle y'_1\rangle \dots \langle y'_t\rangle\langle c\rangle$, where $(a',d',y'_1,\dots,y'_t) = s'_{j''}$. But $s_{j''} = s'_{j''}$, which contradicts $ind_\tau \neq ind'_\tau$.

To conclude, only Case 1 can occur, which gives the desired result of the lemma. □

D.4 The information flow during a list machine's run

In this subsection we take a closer look at the information flow that can occur during a list machine's computation and, using this, we show that only a small number of input positions can be compared during an NLM's run.

Definition 35 (subsequence). A sequence $(s_1,\dots,s_\lambda)$ is a *subsequence* of a sequence $(s'_1,\dots,s'_{\lambda'})$, if there exist indices $j_1 < \dots < j_\lambda$ such that $s_1 = s'_{j_1}, s_2 = s'_{j_2}, \dots, s_\lambda = s'_{j_\lambda}$.

⊣

Definition 36. Let $M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ be an NLM. Let $\gamma = (a,p,d,X)$ be a configuration of M with $X = (x_1,\dots,x_t)^\top$ and $x_\tau = (x_{\tau,1},\dots,x_{\tau,m_\tau})$, for each $\tau \in \{1,\dots,t\}$. Furthermore, let $(i_1,\dots,i_\lambda) \in \{1,\dots,m\}^\lambda$, for some $\lambda \in \mathbb{N}$, be a sequence of input positions.

We say that the sequence $(i_1,\dots,i_\lambda)$ *occurs* in configuration γ, if the following is true: There exists a $\tau \in \{1,\dots,t\}$ and list positions $1 \leq j_1 \leq \dots \leq j_\lambda \leq m_\tau$ such that, for all $\mu \in \{1,\dots,\lambda\}$, the input position i_μ occurs in $ind(x_{\tau,j_\mu})$. ⊣

The following lemma gives a closer understanding of the information flow that can occur during an NLM's run.

Lemma 37 (Merge Lemma). *Let $M = (t,m,I,C,A,a_0,\alpha,B,B_{acc})$ be an (r,t)-bounded NLM, let ρ be a run of M, let γ be a configuration in ρ, and let, for some $\lambda \in \mathbb{N}$, $(i_1,\dots,i_\lambda) \in \{1,\dots,m\}^\lambda$ be a sequence of input positions that occurs in γ.*

Then, there exist t^r subsequences $s_1,\dots,s_{t^r}$ of $(i_1,\dots,i_\lambda)$ such that the following is true, where we let $s_\mu = (s_{\mu,1},\dots,s_{\mu,\lambda_\mu})$, for every $\mu \in \{1,\dots,t^r\}$:

- $\{i_1,\dots,i_\lambda\} = \bigcup_{\mu=1}^{t^r} \{s_{\mu,1},\dots,s_{\mu,\lambda_\mu}\}$, *and*
- *for every $\mu \in \{1,\dots,t^r\}$, s_μ is a subsequence either of $(1,\dots,m)$ or of $(m,\dots,1)$.* ⊣

Proof: By induction on $r' \in \{0,\dots,r\}$ we show that for each configuration that occurs during the r'-th *scan* (i.e., between the $(r'-1)$-st and the r'-th change of a head direction), the above statement is true for $t^{r'}$ rather than t^r.

For the induction start $r' = 0$ we only have to consider M's start configuration. Obviously, every sequence $(i_1,\dots,i_\lambda)$ that occurs in the start configuration, is a subsequence of $(1,\dots,m)$.

For the induction step we note that all that M can do during the r'-th scan is *merge* entries from t different lists produced during the $(r'-1)$-st scan. Therefore, $\{i_1,\dots,i_\lambda\}$ is the union of t sequences, each of which is a subsequence of either $(i_1,\dots,i_\lambda)$ or $(i_\lambda,\dots,i_1)$ (corresponding to a forward scan or a backward scan, respectively), and each of these t subsequences has been produced during the $(r'-1)$-st scan. By induction hypothesis, each of these subsequences is the union of $t^{r'-1}$ subsequences of $(1,\dots,m)$ or $(m,\dots,1)$. Consequently, $(i_1,\dots,i_\lambda)$ must be the union of $t \cdot t^{r'-1}$ such subsequences. □

We are now ready to show that only a small number of input positions can be compared during a list machine's run.

Lemma 38 (Only few input positions can be compared by an NLM).
Let $M = (t,2m,I,C,A,a_0,\alpha,B,B_{acc})$ be an NLM with $2m$ input positions.
Let $\overline{v} := (v_1,\dots,v_m,v'_1,\dots,v'_m) \in I^{2m}$ be an input for M, let ρ be a run of M on input $\overline{v}$, and let $\zeta := skel(\rho)$.
Then, for every permutation φ of $\{1,\dots,m\}$, there are at most

$$t^{2r} \cdot sortedness(\varphi)$$

different $i \in \{1,\dots,m\}$ such that the input positions i and $m+\varphi(i)$ are compared in ζ (i.e., the input values v_i and $v'_{\varphi(i)}$ are compared in ρ).

Proof: For some $\lambda \in \mathbb{N}$ let $i_1,\dots,i_\lambda$ be distinct elements from $\{1,\dots,m\}$ such that, for all $\mu \in \{1,\dots,\lambda\}$, the input positions i_μ and $m+\varphi(i_\mu)$ are compared in ζ. From Definition 33 and 36 it then follows that, for an appropriate permutation $\pi : \{1,\dots,\lambda\} \to \{1,\dots,\lambda\}$, the sequence

$$\iota := \big(i_{\pi(1)}, m+\varphi(i_{\pi(1)}), i_{\pi(2)}, m+\varphi(i_{\pi(2)}), \dots, i_{\pi(\lambda)}, m+\varphi(i_{\pi(\lambda)}) \big)$$

occurs in some configuration in run ρ. From Lemma 37 we then obtain that there exist t^r subsequences $s_1,\dots,s_{t^r}$ of ι such that the following is true, where we let $s_\mu = (s_{\mu,1},\dots,s_{\mu,\lambda_\mu})$, for every $\mu \in \{1,\dots,t^r\}$:

– $\{\, i_1, \ldots, i_\lambda, m+\varphi(i_1), \ldots, m+\varphi(i_\lambda) \,\} \;=\; \bigcup_{\mu=1}^{t^r} \{s_{\mu,1}, \ldots, s_{\mu,\lambda_\mu}\}$, and

– for every $\mu \in \{1, \ldots, t^r\}$, s_μ is a subsequence either of $(1, \ldots, 2m)$ or of $(2m, \ldots, 1)$.

In particular, at least one of the sequences $s_1, \ldots, s_{t^r}$ must contain at least $\lambda' := \lceil \frac{\lambda}{t^r} \rceil$ elements from $\{i_1, \ldots, i_\lambda\}$. W.l.o.g. we may assume that s_1 is such a sequence, containing the elements $\{i_1, \ldots, i_{\lambda'}\}$.
Considering now the set $\{m+\varphi(i_1), \ldots, m+\varphi(i_{\lambda'})\}$, we obtain by the same reasoning that one of the sequences $s_1, \ldots, s_{t^r}$ must contain at least $\lambda'' := \lceil \frac{\lambda'}{t^r} \rceil \geq \frac{\lambda}{t^{2r}}$ elements from $\{m+\varphi(i_1), \ldots, m+\varphi(i_{\lambda'})\}$. We may assume w.l.o.g. that s_2 is such a sequence, containing the elements $m+\varphi(i_1), \ldots, m+\varphi(i_{\lambda''})$.

Let us now arrange the elements $i_1, \ldots, i_{\lambda''}, m+\varphi(i_1), \ldots, m+\varphi(i_{\lambda''})$ in the same order as they appear in the sequence ι. I.e., let $\pi' : \{1, \ldots, \lambda''\} \to \{1, \ldots, \lambda''\}$ be a permutation such that

$$\iota' \;:=\; \big(\, i_{\pi'(1)},\; m+\varphi(i_{\pi'(1)}),\; \ldots,\; i_{\pi'(\lambda'')},\; m+\varphi(i_{\pi'(\lambda'')}) \,\big)$$

is a subsequence of ι.

Since s_1 is a subsequence of ι and a subsequence of either $(1, \ldots, 2m)$ or $(2m, \ldots, 1)$, we obtain that

$$\text{either} \quad i_{\pi'(1)} < i_{\pi'(2)} < \cdots < i_{\pi'(\lambda'')} \quad \text{or} \quad i_{\pi'(1)} > i_{\pi'(2)} > \cdots > i_{\pi'(\lambda'')}.$$

Similarly, since s_2 is a subsequence of ι and a subsequence of either $(1, \ldots, 2m)$ or $(2m, \ldots, 1)$, we obtain that

$$\text{either} \quad m+\varphi(i_{\pi'(1)}) < \cdots < m+\varphi(i_{\pi'(\lambda'')}) \quad \text{or} \quad m+\varphi(i_{\pi'(1)}) > \cdots > m+\varphi(i_{\pi'(\lambda'')}),$$

and therefore,

$$\text{either} \quad \varphi(i_{\pi'(1)}) < \cdots < \varphi(i_{\pi'(\lambda'')}) \quad \text{or} \quad \varphi(i_{\pi'(1)}) > \cdots > \varphi(i_{\pi'(\lambda'')}).$$

In other words, $\big(\varphi(i_{\pi'(1)}), \ldots, \varphi(i_{\pi'(\lambda'')})\big)$ is a subsequence of $\big(\varphi(1), \ldots, \varphi(m)\big)$ that is sorted in either ascending or descending order. According to Definition 19 we therefore have

$$\lambda'' \leq \mathit{sortedness}(\varphi).$$

Since $\lambda'' \geq \frac{\lambda}{t^{2r}}$, we hence obtain that

$$\lambda \leq t^{2r} \cdot \mathit{sortedness}(\varphi),$$

and the proof of Lemma 38 is complete. □

D.5 Proof of Lemma 21

Finally, we are ready for the proof of Lemma 21.

Lemma 21 (Lower Bound for List Machines) — restated.
Let $k, m, n, r, t \in \mathbb{N}$ such that m is a power of 2 and

$$t \;\geq\; 2, \quad m \;\geq\; 24 \cdot (t+1)^{4r} + 1, \quad k \;\geq\; 2m+3, \quad n \;\geq\; 1 + \;(m^2+1) \cdot \log(2k).$$

We let $I := \{0,1\}^n$, identify I with the set $\{0, 1, \ldots, 2^n-1\}$, and divide it into m consecutive intervals $I_1, \ldots, I_m$ each of length $2^n/m$.
Let φ be a permutation of $\{1, \ldots, m\}$ with $\mathit{sortedness}(\varphi) \leq 2\sqrt{m} - 1$, and let

$$\mathscr{I} \quad := \quad I_{\varphi(1)} \times \cdots \times I_{\varphi(m)} \times\; I_1 \times \cdots \times I_m.$$

Then there is no (r,t)-bounded NLM $M = (t, 2m, I, C, A, a_0, \alpha, B, B_{acc})$ with $|A| \leq k$ and $I = \{0,1\}^n$, such that for all $\overline{v} = (v_1, \ldots, v_m, v'_1, \ldots, v'_m) \in \mathscr{I}$ we have:
If $(v_1, \ldots, v_m) \;=\; (v'_{\varphi(1)}, \ldots, v'_{\varphi(m)})$, then $\Pr(M \text{ accepts } \overline{v}) \geq \frac{1}{2}$; otherwise $\Pr(M \text{ accepts } \overline{v}) = 0$. Proof: Suppose for contradiction that M is a list machine which meets the requirements of Lemma 21. We let

$$\mathscr{I}_{eq} \;:=\; \{\, (v_1, \ldots, v_m, v'_1, \ldots, v'_m) \in \mathscr{I} \;:\; (v_1, \ldots, v_m) = (v'_{\varphi(1)}, \ldots, v'_{\varphi(m)}) \,\}.$$

Note that

$$|\mathscr{I}_{eq}| \;=\; \left(\tfrac{2^n}{m}\right)^m.$$

From the lemma's assumption we know that

$$\Pr(M \text{ accepts } \overline{v}) \;\geq\; \frac{1}{2},$$

for every input $\overline{v} \in \mathscr{I}_{eq}$. Our goal is to show that there is some input $\overline{u} \in \mathscr{I} \setminus \mathscr{I}_{eq}$, for which there exists an accepting run, i.e., for which $\Pr(M \text{ accepts } \overline{u}) > 0$. It should be clear that once having shown this, the proof of Lemma 21 is complete.

Since M is (r,t)-bounded, we know from Lemma 31 that there exists a number $\ell \in \mathbb{N}$ that is an upper bound on the length of M's runs. From Lemma 26 we obtain a sequence $\overline{c} = (c_1,..,c_\ell) \in C^\ell$ such that the set

$$\mathscr{I}_{acc,\overline{c}} \;:=\; \{\overline{v} \in \mathscr{I}_{eq} \,:\, \rho_M(\overline{v},\overline{c}) \text{ accepts}\}$$

has size

$$|\mathscr{I}_{acc,\overline{c}}| \;\geq\; \frac{|\mathscr{I}_{eq}|}{2} \;\geq\; \frac{1}{2}\cdot\left(\frac{2^n}{m}\right)^m.$$

Now choose ζ to be the skeleton of a run of M such that the set

$$\mathscr{I}_{acc,\overline{c},\zeta} \;:=\; \{\,\overline{v} \in \mathscr{I}_{acc,\overline{c}} \;:\; \zeta = skel(\rho_M(\overline{v},\overline{c}))\,\}$$

is as large as possible.

CLAIM 2. $\quad |\mathscr{I}_{acc,\overline{c},\zeta}| \;\geq\; \dfrac{|\mathscr{I}_{acc,\overline{c}}|}{(2k)^{m^2}} \;\geq\; \dfrac{1}{2\cdot(2k)^{m^2}}\cdot\left(\dfrac{2^n}{m}\right)^m.$

Proof: Let η denote the number of skeletons of runs of M. From Lemma 32 we know that

$$\eta \;\leq\; \big(2m+k+3\big)^{24\cdot m\cdot(t+1)^{2r+2}+24\cdot(t+1)^r}.$$

From the assumption we know that $k \geq 2m+3$, and therefore

$$\eta \;\leq\; \big(2k\big)^{24\cdot m\cdot(t+1)^{2r+2}+24\cdot(t+1)^r}. \tag{17}$$

From the assumption $m \geq 24\cdot(t+1)^{4r}+1$ we obtain that

$$24\cdot m\cdot(t+1)^{2r+2}+24\cdot(t+1)^r \;\leq\; 24\cdot m\cdot(t+1)^{2r+2}+m \;\leq\; m^2. \tag{18}$$

Altogether, we obtain from (17) and (18) that

$$\eta \quad\leq\quad (2k)^{m^2}.$$

Since the particular skeleton ζ was chosen in such a way that $|\mathscr{I}_{acc,\overline{c},\zeta}|$ is as large as possible, and since the total number of skeletons is at most $(2k)^{m^2}$, we conclude that

$$|\mathscr{I}_{acc,\overline{c},\zeta}| \;\geq\; \frac{|\mathscr{I}_{acc,\overline{c}}|}{(2k)^{m^2}} \;\geq\; \frac{1}{2\cdot(2k)^{m^2}}\cdot\left(\frac{2^n}{m}\right)^m.$$

Hence, the proof of Claim 2 is complete. □

CLAIM 3. *There is an $i_0 \in \{1,..,m\}$ such that the input positions i_0 and $m+\varphi(i_0)$ are not compared in ζ.*

Proof: According to the particular choice of the permutation φ we know that

$$sortedness(\varphi) \;\leq\; 2\cdot\sqrt{m}-1.$$

Due to Lemma 38 it therefore suffices to show that $m > t^{2r}\cdot(2\sqrt{m}-1)$.
From the assumption that $m \geq 24\cdot(t+1)^{4r}+1$ we know that, in particular, $m > 4\cdot t^{4r}$, i.e., $\sqrt{m} > 2\cdot t^{2r}$.
Hence, $t^{2r}\cdot(2\sqrt{m}-1) < \frac{1}{2}\cdot\sqrt{m}\cdot(2\sqrt{m}-1) \leq m$, and the proof of Claim 3 is complete. □

Without loss of generality let us henceforth assume that $i_0 = 1$ (for other i_0, the proof is analogous but involves uglier notation).
Now choose $v_2 \in I_{\varphi(2)}, \ldots, v_m \in I_{\varphi(m)}$ such that

$$\Big|\,\{\,v_1 \in I_{\varphi(1)} \,:\, (v_1,v_2..,v_m,v_{\varphi^{-1}(1)},v_{\varphi^{-1}(2)},..,v_{\varphi^{-1}(m)}) \in \mathscr{I}_{acc,\overline{c},\zeta}\,\}\,\Big|$$

is as large as possible. Then, the number of v_1 such that

$$(v_1,v_2..,v_m,v_{\varphi^{-1}(1)},v_{\varphi^{-1}(2)},..,v_{\varphi^{-1}(m)}) \;\in\; \mathscr{I}_{acc,\overline{c},\zeta}$$

is at least

$$\frac{|\mathscr{I}_{acc,\overline{c},\zeta}|}{\left(\frac{2^n}{m}\right)^{m-1}} \quad\overset{\text{Claim 2}}{\geq}\quad \frac{\left(\frac{2^n}{m}\right)^m}{2\cdot(2k)^{m^2}\cdot\left(\frac{2^n}{m}\right)^{m-1}} \quad\geq\quad \frac{2^n}{2m\cdot(2k)^{m^2}}.$$

From the assumption we know that $n \geq 1+(m^2+1)\cdot\log(2k)$. Therefore,

$$2^n \quad\geq\quad 2\cdot(2k)^{m^2+1} \quad\geq\quad 2\cdot(2k)\cdot(2k)^{m^2} \quad\overset{k\geq m}{\geq}\quad 2\cdot 2m\cdot(2k)^{m^2}.$$

Consequently,

$$\frac{2^n}{2m\cdot(2k)^{m^2}} \quad \geq \quad 2.$$

Thus, there are two different elements $v_1 \neq w_1$ such that for $(w_2,..,w_m) := (v_2,..,v_m)$ we have $\overline{v} := (v_1,..,v_m,v_{\varphi^{-1}(1)},..,v_{\varphi^{-1}(m)}) \in \mathscr{I}_{acc,\overline{c},\zeta}$ and $\overline{w} := (w_1,..,w_m,w_{\varphi^{-1}(1)},..,w_{\varphi^{-1}(m)}) \in \mathscr{I}_{acc,\overline{c},\zeta}$.
Since the run $\rho_M(\overline{v},\overline{c})$ *accepts*, we obtain from Lemma 34 that for the input

$$\overline{u} \;:=\; (v_1,..,v_m,w_{\varphi^{-1}(1)},..,w_{\varphi^{-1}(m)}) \;\in\; \mathscr{I}\setminus\mathscr{I}_{eq},$$

the run $\rho_M(\overline{u},\overline{c})$ has to accept. Therefore, we have found an input $\overline{u}\in\mathscr{I}\setminus\mathscr{I}_{eq}$ with

$$\Pr(M \text{ accepts } \overline{u}) \;>\; 0.$$

This finally completes the proof of Lemma 21. □

E. PROOFS OF LOWER BOUNDS FOR TURING MACHINES

Proof of Corollary 7:
The upper bound is easily obtained when using a result of Chen and Yap [7, Lemma 7] which states that the sorting problem (i.e., the problem of sorting a given sequence of strings) can be solved with two external memory tapes, $O(\log N)$ head reversals, and only constant internal memory space.

The lower bound for the problems CHECK-SORT, SET-EQUALITY, and MULTISET-EQUALITY is stated in Theorem 6. To obtain the according lower bound for the "SHORT" versions of these problems, we reduce the problem CHECK-φ (cf., Lemma 22) to the problems SHORT-CHECK-SORT, SHORT-SET-EQUALITY, and SHORT-MULTISET-EQUALITY (that is, the restriction of these problems to inputs of the form $v_1\#\cdots v_m\#v'_1\#\cdots v'_m\#$, where each v_i and each v'_i is a 0-1-string of length at most $c\cdot\log m$ for some constant $c\geq 2$) in such a way that the reduction can be carried out in $\mathrm{ST}(O(1),O(\log N),2)$. More precisely, we construct a reduction (i.e., a function) f that maps every instance

$$\overline{v} \;:=\; v_1\#\cdots v_m\#v'_1\#\cdots v'_m\#$$

of CHECK-φ to an instance

$$f(\overline{v})$$

of SHORT-CHECK-SORT (respectively, of SHORT-SET-EQUALITY or SHORT-MULTISET-EQUALITY), such that

(1) the string $f(\overline{v})$ is of length $\Theta(|\overline{v}|)$,

(2) $f(\overline{v})$ is a "yes"-instance of SHORT-CHECK-SORT (respectively, a "yes"-instance of SHORT-(MULTI)SET-EQUALITY) if, and only if, $\overline{v}$ is a "yes"-instance of CHECK-φ, and

(3) there is an $(O(1),O(\log N),2)$-bounded deterministic Turing machine that, when given an instance $\overline{v}$ of CHECK-φ, computes $f(\overline{v})$.

It should be clear that the existence of such a mapping f shows that if SHORT-CHECK-SORT (respectively, SHORT-(MULTI)SET-EQUALITY) belongs to the class $\mathrm{RST}(O(r),O(s),O(1))$, for some $s\in\Omega(\log N)$, then also CHECK-φ belongs to $\mathrm{RST}(O(r),O(s),O(1))$. If r and s are chosen according to the assumption of Corollary 7, this would cause a contradiction to Lemma 22. Therefore, SHORT-(MULTI)SET-EQUALITY and SHORT-CHECK-SORT do not belong to the class $\mathrm{RST}(o(\log N),O(\frac{\sqrt[4]{N}}{\log N}),O(1))$.

Now let us concentrate on the construction of the reduction f.
For $i\in\{1,..,m\}$, we subdivide the 0-1-string $v_i\in\{0,1\}^{m^3}$ into $\mu := \lceil\frac{m^3}{\log m}\rceil$ consecutive blocks $v_{i,1},\ldots,v_{i,\mu}$, each of which has length $\log m$ (to ensure that also the last sub block has length $\log m$, we may pad it with leading 0s). In the same way, we subdivide the string v'_i into sub blocks $v'_{i,1},\ldots,v'_{i,\mu}$. For a number $i\in\{1,..,m\}$ we use $\mathrm{BIN}(i)$ to denote the binary representation of $i-1$ of length $\log m$; and for a number $j\in\{1,..,\mu\}$ we use $\mathrm{BIN}'(j)$ to denote the binary representation of $j-1$ of length $3\cdot\log m$.
For every $i\in\{1,..,m\}$ and $j\in\{1,..,\mu\}$ we let

$$\begin{array}{lcl} w_{i,j} & := & \mathrm{BIN}(\varphi(i))\quad \mathrm{BIN}'(j)\quad v_{i,j},\\ w'_{i,j} & := & \mathrm{BIN}(i)\quad \mathrm{BIN}'(j)\quad v'_{i,j},\end{array}$$

for every $i\in\{1,..,m\}$ we let

$$\begin{array}{lcl} u_i & := & w_{i,1}\#w_{i,2}\#\cdots w_{i,\mu}\#,\\ u'_i & := & w'_{i,1}\#w'_{i,2}\#\cdots w'_{i,\mu}\#,\end{array}$$

and finally, we define

$$f(\overline{v}) \;:=\; u_1\cdots u_m\,u'_1\cdots u'_m.$$

Clearly, $f(\overline{v})$ can be viewed as an instance for SHORT-CHECK-SORT or SHORT-(MULTI)SET-EQUALITY, where $m' := \mu \cdot m = \lceil \frac{m^4}{\log m} \rceil$ pairs $w_{i,j}$ and $w'_{i,j}$ of 0-1-strings of length $5 \cdot \log m \leq 2 \cdot \log m'$ are given. Let us now check that the function f has the properties (1)–(3).

ad (1): Every instance $\overline{v}$ of CHECK-φ is a string of length $N = \Theta(m \cdot m^3) = \Theta(m^4)$, and $f(\overline{v})$ is a string of length $N' = \Theta(m^4)$.

ad (2):

$$\begin{aligned} & \overline{v} \text{ is a "yes"-instance of CHECK-}\varphi \\ \iff \quad & (v_1,..,v_m) \;=\; (v'_{\varphi(1)},..,v'_{\varphi(m)}) \\ \iff \quad & (v_{\varphi^{-1}(1)},..,v_{\varphi^{-1}(m)}) \;=\; (v'_1,..,v'_m) \\ \iff \quad & \text{for all } i \in \{1,..,m\},\ (w_{\varphi^{-1}(i),1},..,w_{\varphi^{-1}(i),\mu}) \;=\; (w'_{i,1},..,w'_{i,\mu}). \end{aligned} \tag{19}$$

It is straightforward to see that (19) holds if, and only if, $f(\overline{v})$ is a "yes"-instance of SHORT-(MULTI)SET-EQUALITY. Furthermore, as the list of 0-1-strings in the second half of $f(\overline{v})$ is *sorted* in ascending order, $f(\overline{v})$ is a "yes"-instance of SHORT-CHECK-SORT if, and only if, it is a "yes"-instance of SHORT-(MULTI)SET-EQUALITY.

ad (3): In a first scan of the input tape, a deterministic Turing machine can compute the number m and store its binary representation on an internal memory tape.
Now recall from Remark 20 that the permutation $\varphi = \varphi_m$ is chosen in such a way that for every $i \in \{1,..,m\}$, the binary representation of $\varphi(i)$ is exactly the reverse binary representation of i — and for each particular i, this can be computed on the internal memory tapes. Therefore, during a second scan of the input tape, the machine can produce the string $f(\overline{v})$ on a second external memory tape (without performing any further head reversals on the external memory tapes).

Altogether, the proof of Corollary 7 is complete. □

Proof of Corollary 9:
(a) is an immediate consequence of Theorem 6 and Theorem 8 (a).
The second inequality in (b) follows directly from Theorem 6 and Theorem 8 (b).
The first inequality in (b) holds because, due to Theorem 8 (a), the *complement* of the MULTISET-EQUALITY problem belongs to $\mathrm{RST}(2, O(\log N), 1)$. Since the deterministic $\mathrm{ST}(\cdots)$ classes are closed under taking complements, Theorem 6 implies that the complement of the MULTISET-EQUALITY does not belong to $\mathrm{ST}(O(r), O(s), O(1))$. □

Proof of Corollary 10:
Of course, the CHECK-SORT problem can be solved for input $x_1\#\cdots x_m\#y_1\#\cdots y_m\#$ by (1) sorting $x_1\#\cdots\#x_m$ in ascending order and writing the sorted sequence, $x'_1\#\cdots\#x'_m$ onto the second external memory tape, and (2) comparing $y_1\#\cdots\#y_m$ and the (sorted) sequence $x'_1\#\cdots\#x'_m$ in parallel.

Therefore, if the sorting problem could be solved in *LasVegas*-$\mathrm{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$, i.e., by an $(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$-bounded *LasVegas*-RTM T, then we could solve the CHECK-SORT problem by an $(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$-bounded $(\frac{1}{2}, 0)$-RTM T' which uses T as a subroutine such that T' rejects whenever T answers *"I don't know"* and T' accepts whenever T produces a sorted sequence that is equal to the sequence $y_1\#\cdots\#y_m$. This, however, contradicts Theorem 6. □